SDXML VT2024
Models and languages for semi-structured data and XML

Query languages for SSD and XML

nikos dimitrakas
nikos@dsv.su.se
08-161295

Corresponding reading
Excerpt from Data on the Web
Chapter 2, 3, 9, 10, 11, 13.3, A.3.1, C.2 of the course book
Parts of chapter 30 of Database Systems (Connolly, Begg) 6th edition (chapter 31 in 5th edition)
Articles about XML Query Languages
Compendium about XQuery

Query languages

- **Traverse the data structure**
  - **Path expressions (SSD)**
  - **XPath (XML)**
- **Query the data**
  - **Lorel (SSD)**
  - **XQuery (XML)**
  - **Also on metadata!**
- **Update/change the data/structure**
  - **XQuery Update Facility**

Query languages

Traverse the data structure
Path expressions (SSD)
Xpath(XML)

Query the data
Lorel (SSD)
Xquery(XML)
also on metadata

Update/change data
Xquery update facility

Query languages

- General properties/facilities
  - Querying the database
  - Conditions
  - Aggregations
  - Functions and operation
  - Closed language
- Specific for SSD and XML
  - Traversing the structure
  - Querying the metadata

Query languages
Querying the database
Conditions
Aggregations
Functions and Operation
Closed language

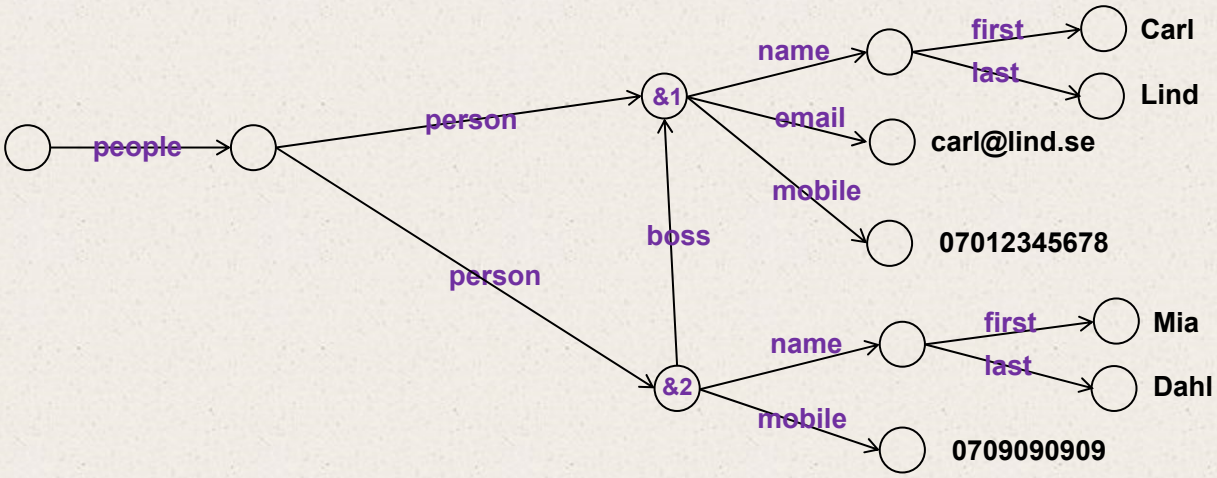
Path expressions

- Traversing the structure
  - Sequence of labels (SSD) or node names (XML)
- The result is a node sequence (or node set)
- A limited query language

SSD Path expressions

- Sequence of labels
  - x.y.z
- Wildcards
  - _
  - _+
  - _*
- Alternatives
  - x|y
- Variables
  - x.L.z (variables in uppercase)
- The result is a node set

Path expressions - Example



- people.person.name.first
- people.person.(email|mobile)
- people._*.mobile
- people._.name

_ is wildcard for any label.
with * - 0 or more label
people.person.mobile and
people.person.boss.mobile

without * - 1 label

Lorel

- Lore Language
  - Lore (Lightweight Object REpository)
- Based on OQL (Object Query Language)
  - OQL is based on (inspired by) SQL
- select … from … where …
- Input: SSD
- Output: SSD

- Support for more DML operations

Sample data

```
db:{person:{name:{first:"Carl", last:"Lind"},
            email:"carl@lind.se",
            mobile:"070111222",
            home:"08151515"},
    person:{name:{first:"Maria", last:"Berg"},
            email:"mb@home.se",
            mobile:"070444555"},
    person:{name:{first:"Peter", nick:"Lightning" last:"Larsson"},
            email:"blixten@gmail.com",
            home:"08789789"},
    person:{name:{first:"Lisa", last:"Lind"},
            email:"lisa@lind.se",
            mobile:"070636363",
            home:"08151515"},
    person:{name:{first:"Mia", nick:"Punky" last:"Persson"},
            mobile:"070199991",
            email:"miap@gmail.com",
            home:"08199991"}
}
```

Lorel - Exampe

```
select name:N
from db.person.name N
```

One iteration per possible N node.

Result:

```
{name:{first:"Carl", last:"Lind"},
name:{first:"Maria", last:"Berg"},
name:{first:"Peter", nick:"Lightning" last:"Larsson"},
name:{first:"Lisa", last:"Lind"},
name:{first:"Mia", nick:"Punky" last:"Persson"}}
```


Lorel - Example

```
select name:N
from db.person P, P.name N
```

Result:

```
{name:{first:"Carl", last:"Lind"},
name:{first:"Maria", last:"Berg"},
name:{first:"Peter", nick:"Lightning" last:"Larsson"},
name:{first:"Lisa", last:"Lind"},
name:{first:"Mia", nick:"Punky" last:"Persson"}}
```

Lorel - Exempel

```
select name:N
from db.person P, P.name N, N.nick S
where S = "Lightning"
```

```
select name:N
from db.person P, P.name N
where N.nick = "Lightning"
```

N.nick is formally a set and should therefore be handled as such:

```
select name:N
from db.person P, P.name N
where "Lightning" in N.nick
```


Lorel - Example, exists

```
select name:N
from db.person.name N
where exists L in N.last : L = "Lind"
```

```
select name:N
from db.person.name N
where "Lind" in N.last
```

```
select name:N
from db.person.name N
where "Lind" = N.last
```

Lorel - Example, nesting

```
select person:(select nickname:S
                      from N.nick S)
from db.person.name N
```

```
select person:{nickname:S}
from db.person.name N, N.nick S
```

Same result?

Lorel - Example, join

```
select name:N
from db.person P, P.name N, db.person P2, P2.name N2
where not (P = P2)
and N2.last = N.last
```

```
select name:N
from db.person P, P.name N
where exists P2 in db.person:
                      not (P = P2)
                      and N.last = P2.name.last
```

Lorel - Example, labels

```
select type:L
from db.person.name N, N.L X
where X = "Lisa"
```

```
select L:V
from db.person P, P.L V
where L in ("mobile", "home")
and "Carl" in P.name.first
```


Lorel - Example, result structure

```
select person:{name:F,
                    contact:{phone:M, mail:E}}
from db.person P,
     P.name N,
     P.email E,
     P.mobile M,
     N.first F
```

```
select person:{name:N.first,
                    contact:{phone:P.mobile, mail:P.email}}
from db.person P, P.name N
```

XPath

- **XPath 1.0**
  - **Limited**
  - **Created together with XSLT 1.0**
  - **Based on the Infoset model**
  - **Uses node sets**
- **XPath 2.0**
  - **More functions, operations, etc.**
  - **Adapted to the XQuery 1.0 model**
  - **Used by XSLT 2.0**
  - **Uses node sequences**
- **XPath 3.0 and 3.1**
  - **Together with XQuery 3.0 and XSLT 3.0**
  - **Dynamic functions and more**

Xpath 1.0
Xpath 2.0
Xpath 3.0 and 3.1

Xpath 1.0
Limited
Created together with XSLT 1.0
Based on Infoset model
Uses node sets

Xpath 2.0
More functions, operations etc
Adapted to Xquery 1.0 model
Used by XSLT 2.0
Uses node sequences

Xpath 3.0 and 3.1
Together with Xquery 3.0 and XSLT 3.0

XQuery 1.0

- **Standard**
- **Model**
- **Query language**
  - **based on (inspired by) SQL, XQL, XML-QL, Lorel, YATL, etc.**
  - **declarative (not procedural)**
  - **includes XPath 2.0**
  - **XQueryX - XQuery in XML syntax**
  - **FLWOR (for let where order by return)**
    - **» Corresponds to SQL SELECT**
  - **transform statements for the rest of the DML statements (from 2011)**
    - **» separate specification**
- **Next version XQuery 3.0**
  - **together with XPath 3.0 and XSLT 3.0**

Xpath 2.0

Standard
Model
Query language
Includes Xpath 2.0

declarative(not procedural)
XqueryX - Xquery in XML syntax

Sample data

```
<Movies>
    <Movie Title="Driven" Year="2001">
        <Actor Name="Burt Reynolds" YearOfBirth="1936" Country="USA"/>
        <Actor Name="Silvester Stallone" YearOfBirth="1946" Country="USA"/>
        <Actor Name="Kip Pardue" YearOfBirth="1976" Country="Canada"/>
        <Director Name="Silvester Stallone" YearOfBirth="1946" Country="USA"/>
        <ProductionCompany>Tri-Star</ProductionCompany>
    </Movie>
    <Movie Title="Antz" Year="1998">
        <Actor Name="Woody Allen" YearOfBirth="1935" Country="USA"/>
        <Actor Name="Silvester Stallone" YearOfBirth="1946" Country="USA"/>
        <Actor Name="Sharon Stone" YearOfBirth="1958" Country="USA"/>
        <Director Name="Eric Darnell" YearOfBirth="1961" Country="Ireland"/>
        <ProductionCompany>Universal</ProductionCompany>
    </Movie>
    <Movie Title="Picking Up the Pieces" Year="2000">
        <Actor Name="Woody Allen" YearOfBirth="1935" Country="USA"/>
        <Actor Name="Sharon Stone" YearOfBirth="1958" Country="USA"/>
        <Actor Name="Alfonso Arau" YearOfBirth="1948" Country="USA"/>
        <Director Name="Eric Darnell" YearOfBirth="1961" Country="Ireland"/>
        <ProductionCompany>Tri-Star</ProductionCompany>
    </Movie>
    …
</Movies>
```


XML Path expressions (XPath)

- **sequence of element names / node names**
  - **elementX/elementY/elementZ**
- **attributes**
  - **@attributeA**
- **Union | and Concatenation ,**
  - **Union | only with nodes**
  - **Concatenation , nodes and values**
- **Intersection and difference (from XPath 2)**
  - **intersect, except**
  - **Only nodes**
- **Axes**
  - **child, parent, ancestor, descendant, following, preceding, …**
  - **abbreviations: . and .. ("current node" and "parent node")**
- **Predicates**
  - **[condition]**

XPath - Examples

- All movies (Movie nodes):
  - /Movies/Movie
  - //Movie

//Movie[@Year=2000]/Director | //Movie[@Year=2003]/Director

- All movies (Movie nodes) from year 2000
  - //Movie[@Year=2000]
- Years of movies by Universal
  - //Movie[ProductionCompany='Universal']/@Year

| - union

- Directors of movies from 2000 and 2003
  - //Movie[@Year=2000]/Director | //Movie[@Year=2003]/Director
- Movie titles with Woody Allen (as actor)
  - //Actor[@Name='Woody Allen']/../@Title
- Root (document node)
  - /

XPath Axes

- child
  - //Movie/child::Director
  - abbreviation: //Movie/Director
- descendant
  - child, or child's child, etc.
  - /Movies/descendant::Director
  - abbreviation: /Movies//Director
- parent
  - //Director/parent::Movie
  - abbreviation: //Director/.. (no guarantee the parent is a Movie node)
  - //Director/parent::*
- ancestor
  - parent, or parent's parent, etc
  - //Director/ancestor::Movies

Xpath Axes

child axis
//Movie/child::Director
abbreviation : //Movie/Director

descendant
/Movies//Director

XPath Axes

- attribute
  - //Movie/attribute::Title
  - abbreviation: //Movie/@Title
- self
  - //Movie/self::Movie
  - abbreviation: //Movie/.
- descendant-or-self
  - //Movie/descendant-or-self::Director
  - //Movie/descendant-or-self::Movie
  - abbreviation: //Movie//Movie
- ancestor-or-self
  - //Director/ancestor-or-self::Director
  - //Director/ancestor-or-self::Movie

XPath Axes

- following-sibling
  - //Movie/Actor/following-sibling::Actor
- preceding-sibling
  - //Movie/Actor/preceding-sibling::Actor
- following
  - nodes that follow, but are not descendants or attributes or namespaces
  - //Movie/following::Actor
- preceding
  - nodes that come before, but are not ancestors or attributes or namespaces
  - //Actor/preceding::Movie
- namespace (deprecated in XPath 2.0)
  - /Movies/namespace::*
  - replaced by functions

XQuery/XPath functions

- Sequence functions:
  - distinct-values(s)
    - » based on string-value()
  - count(s), min(s), max(s), sum(s), avg(s)
  - empty(s), exists(s)
  - reverse(s)

- Node functions:
  - name(n), local-name(n), node-name(n)

XQuery/XPath functions

- String functions:
  - matches(s, regexp)
  - concat(s1,s2)
    - » operator || from XPath 3 and XQuery 3
    - » s1 || s2
  - starts-with(s1,s2), ends-with(s1,s2), contains(s1,s2)
  - substring(s, start), substring(s, start, length)
  - lower-case(s), upper-case(s)
  - replace(s, pattern, replacement)
  - tokenize(s, pattern)

XQuery/XPath functions

doc function - go through filesystem find the file you need and open it.

- **Other functions:**
  - **doc(URI)**
  - **put(n, URI)**
  - **not(e)**
  - **Many date and time functions**
  - **Many numerical functions**
  - **data(ns) - sequence of nodes to sequence of atomic values**
  - **number(n) - the value of the node as a number or NaN**
  - **string(n) - the value of the node as a string**
  - **current-time(), current-date(), current-dateTime()**
  - **position() - the node's position in the current sequence**
  - **last() – returns the position of the last node in the current sequence (the size of the current sequence)**

not(e) - function that negates an expression

Many date and time functions
Many numerical functions

data(ns) - sequence of nodes to sequence of atomic values

number(n) value of node as a number or NaN

string(n) - value of node as a string

Many date and time functions

data(ns) - sequence of nodes to sequence of atomic values

string(n) - value of node as string.

position() - node's position in the current sequence

last() - number of the last node.

The put() function in XQuery and XPath, as mentioned in the slide you provided, is used to save changes made to a node within a document. When using put(n, URI), n represents the node (or document node) that you want to save, and URI is the Uniform Resource Identifier where the node should be saved.

XQuery functions

- **Wildcards (kind tests)**
  - **node() (all nodes other than attributes and namespaces)**
  - **text()**
  - **comment()**
  - **processing-instruction()**
  - **element(), ***
  - **attribute(), @***
  - **document-node()**

node()
text()
comment()
processing-instruction()
element(),*
attribute(),@*
document-node()

Wildcards(kind tests)

XQuery/XPath operators

- +, -, *, div, mod
- =, !=, >, <, <=, >= (general comparisons)
- eq, ne, lt, le, gt, ge (value comparisons)
- or, and
- is, >>, << (node comparisons)

- to (create sequence)
  - 1 to 5 = (1,2,3,4,5)

- union, intersect, except
  - require node sequences
  - use the node identities

XQuery/XPath operators

- / (Path operator)
  - removes duplicates of nodes
  - uses the node identities

- , (Comma operator) and () (sequence construction)
  - (1,2,3,4)
  - ((1,2), 3, (4,5)) becomes (1,2,3,4,5)
  - () empty sequence
  - (//Actor, //Director)

XPath predicates

- Conditions that filter a sequence
  - Only items that satisfy the condition remain
  - Specified inside [ ]
  - Expressions must be true or non empty

```
//Movie[ProductionCompany="Tri-Star"]
//Movie[ProductionCompany]
//Movie[position()=3] (the third movie) Abbreviation: //Movie[3]
//Movie[Actor/@Name="Woody Allen" or @Title="Catwoman"]
//Movie[@Title eq "Catwoman"]/Actor[@Country="USA"]
//Movie[Actor/@Country="USA"][Actor/@Country="Canada"]
same as
//Movie[Actor/@Country="USA" and Actor/@Country="Canada"]
//Movie[count(Actor[@Country="USA"])>2]
//Movie/Actor[last()-1]
(//Movie/Actor)[last()-1]
```


XQuery - FLWOR

- For
  - Loops through sequences (nodes or values)
- Let
  - Assignments
- Where
  - Conditions
- Order By
  - Sorting (affects the result)
  - ascending (default) or descending
- Return
  - Construction of the result

For - Loops through sequences(nodes or values)

Let - Assignments

Where - conditions

For,Let,Where,Order By,Return

For - Loops through sequences (nodes or values)

Let -Assignments
Where - Conditions

Order By - Sorting in ascending or descending

Return - Construction of result

XQuery

- FLWOR statements may be nested.
- No clause is compulsory.
- for and let may appear multiple times in random order (before the where clause).
- XPath expressions may be used in all clauses.
- The result can be well-formed XML, but it does not have to.
  - the result can be anything that is supported by the XQuery model, meaning a sequence of nodes and/or values
- The function doc() can be used to define a source/context (an XML document). Otherwise, the execution environment can be used to configure the source/context.

function doc() can be used to define a source context. Else the execution environment can be used to configure the source context

XQuery

- Variables start with $:
  - for $a in //Movie/Actor
  - let $n := $a/@Name
- Sequences:
  - for $x in (1, 2, 3)
    - » same as for $x in 1 to 3
  - let $y := (1, 2, 3)
- Evaluation of expressions:
  - Expression to be evaluated inside { }:
  - <result>{$x*3}</result>

Xquery
Variables start with $
for $a in //Movie/Actor
let $n:=$a/@Name

n= Actor's name

XQuery - Computed Constructors

- element
  - element name value:

```
let $a := "a", $b  := 2
return <x>{element {$a} {$b}}</x>
```

    - » gives <x><a>2</x>
- attribute
  - attribute name value:

```
let $a := "a", $b  := 2
return <x>{attribute {$a} {$b}}</x>
```

    - » gives <x a="2"/>
- comment
  - comment value
  - comment {"Ahoy"}
    - » gives <!--Ahoy-->

XQuery - Computed Constructors

- processing-instruction
  - processing-instruction name value
  - processing-instruction Say {"Hello"}
    - » gives <?Say Hello?>
- text
  - text value
  - text {"Howdy"}
    - » creates a text node
    - » same result as "Howdy"
- document
  - document content
  - document {comment {"The best movies"}, element Movies {…}}

XQuery - Conditional expressions

- if-then-else

```
for $a in (1 to 5)
return if ($a mod 2 = 0)
            then <even>{$a}</even>
            else <odd>{$a}</odd>
```

```
<odd>1</odd>
<even>2</even>
<odd>3</odd>
<even>4</even>
<odd>5</odd>
```


XQuery - Quantified expressions

- some

```
for $a in //Movie
where some $b in $a/Actor/@Country satisfies string($b) = "Austria"
return  $a
```

- every

```
for $a in //Movie
where every $b in $a/Actor/@Country satisfies string($b) = "USA"
return  $a
```

XQuery - Nested statements

- **The result of a statement becomes a source for another statement:**

```
for $x in distinct-values (for $a in (1 to 6), $b in (1 to 6)
                           return <sum>{$a + $b}</sum>)
return <unique>{$x}</unique>
```

The result of a statement becomes a source for another statement.

- **The result of a statement is assigned to a variable:**

```
for $x in (1 to 6)
let $content := for $a in (1 to 6)
                return element Plus {attribute value {$a}, $x+$a}
return element Number {attribute value {$x}, $content}
```

The result of a statement is assigned to a variable

<Number value="1"><Plus value="1">2</Plus><Plus value="2">3</Plus><Plus value="3">4</Plus><Plus value="4">5</Plus><Plus value="5">6</Plus><Plus value="6">7</Plus></Number>
<Number value="2"><Plus value="1">3</Plus><Plus value="2">4</Plus><Plus value="3">5</Plus><Plus value="4">6</Plus><Plus value="5">7</Plus><Plus value="6">8</Plus></Number>
<Number value="3"><Plus value="1">4</Plus><Plus value="2">5</Plus><Plus value="3">6</Plus><Plus value="4">7</Plus><Plus value="5">8</Plus><Plus value="6">9</Plus></Number>
<Number value="4"><Plus value="1">5</Plus><Plus value="2">6</Plus><Plus value="3">7</Plus><Plus value="4">8</Plus><Plus value="5">9</Plus><Plus value="6">10</Plus></Number>
<Number value="5"><Plus value="1">6</Plus><Plus value="2">7</Plus><Plus value="3">8</Plus><Plus value="4">9</Plus><Plus value="5">10</Plus><Plus value="6">11</Plus></Number>
<Number value="6"><Plus value="1">7</Plus><Plus value="2">8</Plus><Plus value="3">9</Plus><Plus value="4">10</Plus><Plus value="5">11</Plus><Plus value="6">12</Plus></Number>

The result of a statement becomes a source for another statement

Cannot be accessed through an axis

Namespace aliases not available

XQuery - Namespaces

- **Cannot be accessed through an axis**
- **Namespace aliases are not available**
- **To work with namespaces and aliases, they need to be declared in the XQuery prolog:**
  - **declare namespace alias = "URI";**
  - **declare default element namespace "URI"**

```
declare namespace aaa = "URI-1";
declare default element namespace "URI-2";
element Result {for $x in (1 to 5)
                return element aaa:Number {attribute value {$x}}}
```

aaa-alias

```
<Result xmlns="URI-2">
  <aaa:Number xmlns:aaa="URI-1" value="1"/>
  <aaa:Number xmlns:aaa="URI-1" value="2"/>
  <aaa:Number xmlns:aaa="URI-1" value="3"/>
  <aaa:Number xmlns:aaa="URI-1" value="4"/>
  <aaa:Number xmlns:aaa="URI-1" value="5"/>
</Result>
```

XQuery prolog

- **Declare**
  - **functions**
  - **variables**
  - **namespaces**
  - **collation**
  - **sorting**
  - **etc.**

functions
variables
namespaces
collation
sorting
etc

XQuery 3.0 and 3.1

- **News**
  - **group by clause**
  - **the different clauses may appear in a more flexible order**
  - **switch expression**
  - **count clause**
  - **try/catch expression**
  - **dynamic and inline functions**
  - **computed constructor for namespace**
  - **many new functions, among others for mathematical operations**
  - **XQuery node test functions formally part of XPath**
  - **and much more**

News
group by clause
Different clauses may appear in more flexible order
switch expression
count clause
try/catch expression

XQuery 3 - group by

- Groups the iterations so that
  - the return clause is executed once per group
  - the iteration variables become sequences containing all the values of the relevant iterations

1 even=False
2 even =True
3 even=False
4 even=True
5 even =False

```
for $x in 1 to 5
let $even := ($x mod 2) = 0
group by $even
return element Numbers {attribute even {$even}, $x}
```

one row per group if u have group by

Result:

```
<Numbers even="false">1 3 5</Numbers>
<Numbers even="true">2 4</Numbers>
```


XQuery 3 - group by

- can group on one or more variables
- the grouping variables can be declared earlier or in the group by clause
  - Note! If a variable is reused, its content is overwritten

```
for $x in (1,1,2,3,2,1,2,3,2)
group by $x
return element Number {
        attribute times {count($x)}, $x}
```

Result:

```
<Number times="1">1</Number>
<Number times="1">2</Number>
<Number times="1">3</Number>
```

```
for $x in (1,1,2,3,2,1,2,3,2)
group by $y := $x
return element Number {
        attribute times {count($x)}, $y}
```

Result:

```
<Number times="3">1</Number>
<Number times="4">2</Number>
<Number times="2">3</Number
```

XQuery 3 - Mixed clauses

```
for $x in (1,1,2,3,2,1,2,3,2,4,5,3)
where $x < 5
group by $v := $x
where count($x) > 1
let $s := sum($x)
return <Number value="{$v}" sum="{$s}" />
```

1,1,2,3,2,1,2,3,2,4,5,3

group by $v:=$x

Result:

```
<Number value="1" sum="3"/>
<Number value="2" sum="8"/>
<Number value="3" sum="9"/>
```

Start with a let or a for and finish with a return. The rest is free!

XQuery 3 - switch

- **Instead of nested if-then-else**
  - many case
  - one default

```
for $x in 0 to 3
return element {switch ($x)
                  case 0 return "Zero"
                  case 1 return "One"
                  default return "Many"} {$x}
```

Result:

```
<Zero>0</Zero>
<One>1</One>
<Many>2</Many>
<Many>3</Many>
```

XQuery 3 - count (clause)

- **Counts the iterations that reach it (the count clause)**

x in 4 to 10
4,5,6,7,8,9,10
count $loop

```
for $x in 4 to 10
count $loop
let $nested := let $a := $x count $i return $i
where $x mod 2 = 0
count $correct
return <Number x="{$x}" loop="{$loop}" correct="{$correct}" nested="{$nested}"/>
```

Number x="4",6,8,10

Result:

```
<Number x="4" loop="1" correct="1" nested="1"/>
<Number x="6" loop="3" correct="2" nested="1"/>
<Number x="8" loop="5" correct="3" nested="1"/>
<Number x="10" loop="7" correct="4" nested="1"/>
```


XQuery 3 - dynamic functions

- **Functions may be handled as values and assigned to variables**
- **All functions can be referred to with their name and cardinality**
  - **Example: The function count that takes one parameter is count#1**

$f:=(upper-case#1,lower-case#1)

dynamic functions
functions may be handled as values and assigned to variables.

```
let $f := (upper-case#1, lower-case#1)
for $x in 1 to 4
return element Result {$f[$x mod 2 + 1]("STockHOLm")}
```

Result:

$f[0]("Stockholm")

all functions can be referred to with their name and cardinality
ex - function count that takes one parameter is count#1

```
<Result>stockholm</Result>
<Result>STOCKHOLM</Result>
<Result>stockholm</Result>
<Result>STOCKHOLM</Result>
```

$f:=(upper-case #1,lower - case#1)
1 mod 2 +1 = 2("STockHOLm")
1+1 =2 ("STockHOLm")
2("STOCKHOLM) - lower case
2mod2 +1
1("Stockholm") = upper case

XQuery 3 - inline functions

- **Functions may be defined inline (without naming them)**

$p in 1 to 4

```
let $f := function($a) {sum(let $as := string($a)
                            for $p in 1 to string-length($as)
                            return xs:integer(substring($as, $p, 1))) }
for $x in (1999, 5005005, 123456789)
return element Result {attribute input {$x}, $f($x)}
```

$f($x) = $f(1999)
=$f sum(as=

Result:

```
<Result input="1999">28</Result>
<Result input="5005005">15</Result>
<Result input="123456789">45</Result>
```


XQuery 3 - higher-order functions

- **for-each (sequence, function)**
  - **Calls a function for all the items in a sequence**

```
sum(for-each(1 to 4, function($a) {$a*$a}))      Returns 30
sum(for-each(-4 to 3, abs#1))                    Returns 16
```

- **filter (sequence, function)**
  - **Returns only the sequence items that make the function return true**

filter(("Maria","Lisa","Rebecka","Ylva"))

```
filter(("Maria", "Lisa", "Rebecka", "Ylva", "Evelina"),
       function($s) {contains($s, "e")})
```

function($s){contains($s,"e")}

Returns the sequence ("Rebecka", "Evelina")

filter - returns only sequence items that make functions return True

XQuery 3 - Computed Constructors

- namespace
  - namespace prefix URI
  - namespace sdxml {"http://ns.dsv.su.se/SDXML"}
  - namespace {""} {"http://ns.dsv.su.se/SDXML"}
  - Namespaces created in this way may not be used directly
  - Use a declaration in the XQuery prolog instead, or hardcode them as xmlns "attributes"

XQuery Update Facility

- According to XQUF 1.0:
- transform statement
  - copy clause
  - modify clause
    - » delete expression
    - » insert expression
    - » rename expression
    - » replace expression
  - return clause
- Supports nested FLWOR
  - that return update expressions

before a certain node

- keywords for insert
  - » node or nodes
  - » before
  - » after
  - » as first into
  - » as last into
  - » into
- keywords for delete
  - node or nodes
- keywords for replace
  - » (value of) node … with …
- » keywords for rename
  - » node … as …

XQuery - transform

- copy
  - create a copy of an XML document
- modify
  - one or more expressions that update/change the copy
- return
  - usually the copy, but result construction is possible

XQuery - transform - insert

```
copy $x := <root><a>456</a><a>789</a></root>
modify insert node <a>123</a> as first into $x
return $x
```

```
<root>
        <a>123</a>
        <a>456</a>
        <a>789</a>
</root>
```

XQuery - transform - insert

```
copy $x := <root><a>456</a><a>789</a></root>
modify insert node <a>123</a> before $x/a[text() = 789]
return $x
```

copy $x:= <root> <a> 456<a>

before $x/ a[text() = 789]

```
<root>
        <a>456</a>
        <a>123</a>
        <a>789</a>
</root>
```


XQuery - transform - insert

```
copy $x := <root><a>456</a><a>789</a></root>
modify insert node attribute c {5} into $x/a[text() = 789]
return $x
```

```
<root>
        <a>456</a>
        <a c="5">789</a>
</root>
```

XQuery - transform - delete

```
copy $x := <root><a>456</a><a>789</a></root>
modify delete node $x/a[text() = 789]
return $x
```

```
<root>
        <a>456</a>
</root>
```


XQuery - transform - delete

```
copy $x := <root><a>456</a><a>789</a></root>
modify delete node $x/a
return $x
```

```
<root/>
```

XQuery - transform - replace

```
copy $x := <root><a>456</a><a>789</a></root>
modify replace node $x/a[text() = 789] with <b>123</b>
return $x
```

```
<root>
        <a>456</a>
        <b>123</b>
</root>
```


XQuery - transform - replace

```
copy $x := <root><a>456</a><a>789</a></root>
modify replace value of node $x/a[text() = 789] with 123
return $x
```

```
<root>
        <a>456</a>
        <a>123</a>
</root>
```

XQuery - transform - replace

```
copy $x := <root><a b="ccc">456</a><a>789</a></root>
modify replace node $x/a[1]/@b with attribute f {"ddd"}
return $x
```

modify replace node $x/a[1]/@b with attribute f
replace b with attribute f

```
<root>
        <a f="ddd">456</a>
        <a>789</a>
</root>
```


XQuery - transform - rename

```
copy $x := <root><a>456</a><a>789</a></root>
modify rename node $x/a[2] as "b"
return $x
```

rename $x/a[2] as "b"

```
<root>
        <a>456</a>
        <b>789</b>
</root>
```

XQuery - transform - more, loop

```
copy $x := <root><a>456</a><a>789</a></root>
modify for $a in $x/a
        return rename node $a as "b"
return $x
```

```
for $a in $x/a
return rename node $a as "b"
```

rename all a nodes as b

modify with a loop

```
<root>
      <b>456</b>
      <b>789</b>
</root>
```


XQuery - transform - more, list

```
copy $x := <root><a>456</a><a>789</a></root>
modify (rename node $x/a[1] as "b",
      rename node $x/a[2] as "c")
return $x
```

```
<root>
      <b>456</b>
      <c>789</c>
</root>
```

XQuery - transform - more, list

```
copy $x := <root><a>456</a><a>789</a></root>
modify (rename node $x/a[1] as "b",
        insert node attribute f {"200"} into $x/a[1],
        replace value of node $x/a[1] with "123")
return $x
```

```
<root>
        <b f="200">123</b>
        <a>789</a>
</root>
```

$x : = <456> <789>
xquery - transform - more,list
modify (rename node $x/a[1] as "b",

inside modify several sentences

Note! The element b does not exists until after the execution of all the expressions (which is done at the end).

XQuery Update Facility

- According to XQUF 3.0:
- Copy-modify statement
  - As before:
  - copy clause
  - modify clause
  - return clause
- Transform-with statement
  - … transform with {expressions}
  - Possible expressions
  - insert
  - delete
  - replace
  - rename
- Expressions in return
  - insert
  - delete
  - replace
  - rename
- "Updating expressions"
  - Modify the current node

transform with - insert,delete,replace,rename

XQuery - transform with

```
<root><a>456</a><a>789</a></root>
transform with {insert node <a>123</a> as first into .}
```

Same effect as

```
copy $x := <root><a>456</a><a>789</a></root>
modify insert node <a>123</a> as first into $x
return $x
```

```
<root>
        <a>123</a>
        <a>456</a>
        <a>789</a>
</root>
```


XQuery - transform with

```
let $x := <root><a>456</a><a>789</a></root>
return $x transform with {
        insert node <a>123</a> as first into .,
        rename node ./a[2] as "b"
}
```

```
<root>
        <a>123</a>
        <a>456</a>
        <b>789</b>
</root>
```

Note! The second a is based on the original/unmodified structure, since no changes have been applied yet.

XQuery - Updating expressions

```
rename node //Movie[1] as "Picture"
```

The node is modified and nothing is returned.

```
for $m in //Movie
return rename node $m as "Picture"
```

All Movie elements are changed to Picture and nothing is returned.

What is returned, is actually one or more "pending updates". Nothing visible.

XQuery Update Facility - Support

- Supported in BaseX
  - According to XQuery Update Facility 3.0
  - Copy-modify
  - Transform-with
  - Updating expressions
  - Does not change the files, only the in-memory copy.
- Supported in DB2 and Oracle (from version 12)
  - According to XQuery Update Facility 1.0
    - » Copy-modify
  - Some syntactical differences
- Supported in some other products

What to do next

- Quiz about XQuery & XPath (Quiz 5)
- XQuery (compendium Querying XML Data with XQuery)
  - Contains more examples
  - Contains information about BaseX and XQuisitor
- Lesson exercises (1 & 2)
- Seminar exercises (Assignment 1)
  - Wait until after the lessons