SDXML VT2026
Models and languages for semi-structured data and XML

Query languages for SSD and XML
Lorel, XPath

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

Corresponding reading
Excerpt from Data on the Web
Chapter 2, 3, 9 of the course book
Parts of chapter 30 of Database Systems (Connolly, Begg) 6th edition (chapter 31 in 5th edition)

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

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

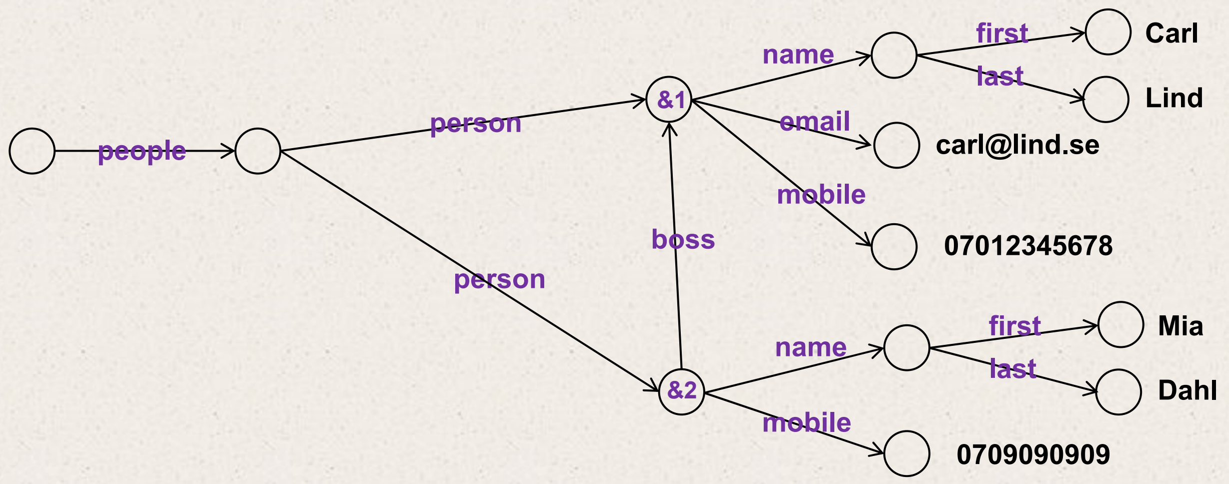
Path expressions

- 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

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 - Example

```
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 - Example

```
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

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

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

- 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
- 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

- 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)

XQuery functions

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

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]

What to do next

- Quiz about XPath (Quiz 5)
- Lesson exercises (Lesson 1)