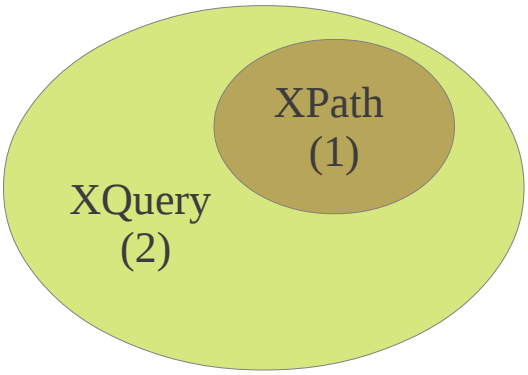
XML Datenbanken

Quickstart für die Anfragesprachen XPath und XQuery

Harald Lang

3. Juli 2014



(1) Pfadausdrücke mit XPath

- *Adressieren* von Elementen innerhalb eines hierarchischen XML-Dokuments
- Setzt sich auch einem oder mehreren *Lokalisierungsschritten* zusammen (getrennt durch "/")

$$\underbrace{\texttt{Achse} :: \texttt{Knotentest}\underbrace{[\texttt{Prädikat 1}][\texttt{Prädikat 2}][...]}_{optional}}_{Lokalisierungschritt}\ \underbrace{/.../.../...\qquad .../}_{weitere\ Lokalisierungschritte}$$

Beispiel 1: Mehrere Lokalisierungsschritte

```
doc('uni')/child::Universität/child::Fakultäten/child::FakName
```

Liefert:

```
<FakName>Theologie</FakName>
<FakName>Physik</FakName>
...
```

Beispiel 2: Lokalisierung über die descendant-or-self Achse

```
doc('uni')/descendant-or-self::FakName
```

Liefert alle XML Elemente mit dem Namen "FakName" unabhängig davon, wo diese in der Hierarchie eingehängt sind.

Kurzform: `doc('uni')//FakName`

Beispiel 3: (Selektions-) Prädikat auf ein XML Element

```
doc('uni')//ProfessorIn[child::Name='Sokrates']
```

Liefert das XML Element des Professors mit dem Namen Sokrates.

Kurzform: `doc('uni')//ProfessorIn[Name='Sokrates']`

Beispiel 4:(Selektions-) Prädikat auf ein XML Attribut

```
doc('uni')//ProfessorIn[attribute::PersNr='P2125']
```

Liefert das XML Elemente des Professors mit der Personalnummer 'P2125'.

Kurzform: `doc('uni')//ProfessorIn[@PersNr='P2125']`

Beispiel 5: Verwendung der rückwärts gerichteten parent Achse

```
doc('uni')//ProfessorIn[Name='Sokrates']/parent::Fakultät/FakName
```

Liefert den Namen der Fakultät, an der Sokrates lehrt.

Kurzform: `doc('uni')//ProfessorIn[Name='Sokrates']/../FakName`

(2) XQuery

- FLWOR-Ausdrücke (sprich *flower*)

`for $var in (...)`	bindet die Variable sukzessive an alle Elemente der Liste
`let $n := (...)`	bindet die Variable. $n enthält die vollständige Liste
`where`	(Selektions-) Bedingung
`order by`	Sortierung
`return`	erzeugt die Ausgabe für den aktuellen Schleifendurchlauf

- ermöglicht die Erzeugung beliebiger XML Elemente (beachte: Die Rückgabe eines XPath Ausdrucks erzeugt i.d.R. kein wohlgeformtes XML, meist fehlt das Wurzelelement)
- XPath wird verwendet um in XQuery Variablen zu binden

Beispiel 1: Ein einfaches XQuery Programm

```
<Fakultaeten>
    {
    for $f in doc('uni')//FakName
    return $f
    }
</Fakultaeten>
```

Liefert ein wohlgeformtes XML mit "Fakultaeten" als Wurzel:

```
<Fakultaeten>
  <FakName>Theologie</FakName>
  <FakName>Physik</FakName>
  <FakName>Philosophie</FakName>
</Fakultaeten>
```

Das Wurzelelement "Fakultaeten" wird direkt in Anfrage definiert. So lassen sich beliebige XML Strukturen erzeugen. Die geschweiften Klammern leiten einen FLWOR-Ausdruck ein, dieser wird in der Datenbank zur Ausführung gebracht. Die XML-Tags, welche ausserhalb der geschweiften Klammern stehen, gehen unverändert in die Ausgabe über. XQuery Programmcode kann beliebig tief geschachtelt werden, daher ist immer auf die korrekte Klammerung zu achten.

Beispiel 2: Alternative XQuery Syntax (nicht in der Vorlesung behandelt, darf aber in der Klausur verwendet werden)

```
<VorlesungsVerzeichnisNachFakultaet>
  {for $f in doc('uni')/Universitaet/Fakultaeten/Fakultaet
   return
```

```
    <Fakultaet>
      <FakultaetsName>{$f/FakName/text()}</FakultaetsName>
    </Fakultaet>}
</VorlesungsVerzeichnisNachFakultaet>
```

Die Elementnamen "`VorlesungsVerzeichnisNachFakultaet`", "`Fakultaet`" und "`FakultaetsName`" treten als Konstanten in der Anfrage mehrfach auf; einmal als öffnendes Tag und einmal als schließendes Tag. Diese Tags werden *direct constructors* genannt. Alternativ können *computed constructors* verwendet werden (siehe unten). Dabei werden die schließenden Tags überflüssig und die Klammerung wird übersichtlicher. Zudem erlauben Computed constructors im Gegensatz zu den oben gezeigten *direct constructors* die Elementnamen dynamisch zur Laufzeit zu setzen.

```
element VorlesungsVerzeichnisNachFakultaet {
  for $f in doc('uni')/Universitaet/Fakultaeten/Fakultaet
  return element Fakultaet {
    element FakultaetsName {
      $f/FakName/text()
    }
  }
}
```

Noch ein Beispiel zur Element-Konstruktion mit Attributen (ohne XQuery Code):

```
element Vorlesung {
  attribute VorlNr { "2031" },
  attribute Lehrstuhl { "3" },
  "Einsatz und Realisierung von Datenbanksystemen"
}
```

Beispiel 3: FLWOR-Ausdruck - nur FOR und RETURN, sowie text() funktion

```
<Fakultaeten>
  {
    for $f in doc('uni')//FakName
    return <Fakultaet> { $f/text() } </Fakultaet>
  }
</Fakultaeten>
```

Anders als in Beispiel 1, wird der Name der Fakultät in einem "Fakultaet" Element ausgegeben. Dazu muss auf den Inhalt des "FakName" Elements zugegriffen werden. Dies geschieht mit der Funktion `text()`[1].

[1] Liste aller verfügbaren Funktionen in XQuery: http://www.w3schools.com/xpath/xpath_functions.asp

Beispiel 4: FLWOR-Ausdruck - mit FOR, WHERE und RETURN

```
<Fakultaeten>
  {
    for $f in doc('uni')//FakName
    where $f/text() = 'Physik'
    return <Fakultaet> { $f/text() } </Fakultaet>
  }
</Fakultaeten>
```

Wie Beispiel 3, mit dem Unterschied, dass hier eine WHERE-Bedingung nur die Physik Fakultät selektiert.

Beispiel 5: Vollständiger FLWOR-Ausdruck

```
<ProfessorenLehrbelastung>
  {
    for $p in doc('uni')//ProfessorIn
    let $v := $p/Vorlesungen/Vorlesung
    where count($v) > 1
    order by sum($v/SWS)
    return
       <ProfessorIn>
        <Name> { $p/Name/text() } </Name>
        <Lehrbelastung> { sum($v/SWS) } </Lehrbelastung>
      </ProfessorIn>
  }
</ProfessorenLehrbelastung>
```

Der XPath Ausdruck `doc('uni')//ProfessorIn` lokalisiert alle "ProfessorIn"-Elemente und gibt diese als Liste zurück. Über diese Liste wird mit Hilfe des `for`-Konstrukts iteriert. Dabei wird der Schleifenvariablen `$p` das aktuelle Element aus der Liste zugewiesen.

Der zweite XPath Ausdruck `$p/Vorlesungen/Vorlesung` lokalisiert alle "Vorlesung"-Elemente in `$p`, welche der Variablen `$v` zugewiesen werden. `$v` enthält somit eine *Liste* von "Vorlesung"-Elemente.

Die `WHERE`-Bedingung fordert, dass mindestens ein Element in der Liste enthalten ist. `count()` ist eine globale Funktion, welche als Argument eine Liste erwartet. Der Rückgabewert ist die Länge der Liste bzw. die Anzahl der Elemente. Es werden also nur ProfessorInnen ausgegeben, die mindestens eine Vorlesung halten. Im `return` Statement wird das Ausgabeformat festgelegt. Das Element "Lehrbelastung" enthält dabei die Summe der SWS, diese wird mit Hilfe der Aggregatsfunktion `sum()` gebildet.

Die `order by`-Klausel sortiert die Ausgabe (aufsteigend) nach der Lehrbelastung.

Beispiel 6: Dereferenzierung mit id()

```
<StudentenLernbelastung>
  {
    for $s in doc('uni2')//Student
    let $v := doc('uni2')//id($s/hoert/@Vorlesungen)
    where count($v) > 1
    order by sum($v/SWS)
    return
       <StudentIn>
        <Name> { $s/Name/text() } </Name>
        <Lernbelastung> { sum($v/SWS) } </Lernbelastung>
      </StudentIn>
  }
</StudentenLernbelastung>
```

Elemente die über das spezielle Attribut `ID` verfügen, können direkt mit Hilfe der `id()` Funktion dereferenziert werden. Der XPath Ausdruck `doc('uni2')//id('V4052')` liefert z. B. die Vorlesung "Logik" zurück. Innerhalb eines FLWOR-Ausdrucks kann somit direkt auf andere Elemente zugegriffen werden, ohne dass durch die Hierarchie des XML Dokuments navigiert werden muss. Im Beispiel oben, wird über die Studenten iteriert und in jeder Iteration werden alle Vorlesungen lokalisiert, die der Student hört.

Beispiel 6: Dereferenzierung mit tokenize()

```
<StudentenLernbelastung>
  {
    for $s in doc('uni')//Student

    (: Inhalt des Attributs @Vorlesungen in eine Liste mit
       Vorlesungsnummern zerlegen; und diese Liste an die
       Variable $vorlNrListe binden. :)
    let $vorlNrListe := tokenize($s/hoert/@Vorlesungen, "␣")

    (: Alle Vorlesungsnummern dereferenzieren in einer weiteren
       for-Schliefe. Das for-return-Konstrukt liefert wiederum
       eine Liste, die jedoch die vollständigen 'Vorlesung'-Elemente
       enthält. Rückgabewert ist also eine Liste von Elementen. :)
    let $v := for $vorlNr in $vorlNrListe
              return doc('uni')//Vorlesung[ @VorlNr = $vorlNr ]

    (: Der Rest bleibt unverändert. :)
    where count($v) > 1
    order by sum($v/SWS) descending
```

```
    return
       <StudentIn>
        <Name> { $s/Name/text() } </Name>
        <Lernbelastung> { sum($v/SWS) } </Lernbelastung>
      </StudentIn>
  }
</StudentenLernbelastung>
```