

# XSLT, Teil 4

Udo Kelter

02.05.2019

## **Zusammenfassung dieses Lehrmoduls**

Die Berechnung von Verbunden mit Variablen ist relativ ineffizient; als effizientere Alternative bieten sich manuell verwaltete Sekundärindexe an. Dieses Lehrmodul stellt XSLT-Sekundärindexe vor. Mit diesen kann man auch Gruppierungen anhand von in der Eingabe vorkommenden Datenwerten realisieren.

## **Vorausgesetzte Lehrmodule:**

- obligatorisch:
- XPath
  - XSLT, Teil 1 (Stichworte)
  - XSLT, Teil 2 (Stichworte)
  - XSLT, Teil 3 (Stichworte)

**Stoffumfang in Vorlesungsdoppelstunden:** 0.5

# Inhaltsverzeichnis

<b>1</b>	<b>Sekundärindexe</b>	<b>3</b>
1.1	Anlegen eines Sekundärindexes . . . . .	3
1.2	Benutzen eines Sekundärindexes . . . . .	4
<b>2</b>	<b>Verbundbildung mit Index</b>	<b>4</b>
2.1	Verbundbildung als “Anreicherung” eines Elements . . . . .	5
<b>3</b>	<b>Gruppierung und Aggregation mit Index</b>	<b>7</b>
3.1	Duplikateliminierung . . . . .	7
3.2	Bewertung von Indexen . . . . .	9
<b>4</b>	<b>Softwaretechnische Beurteilung von XSLT</b>	<b>9</b>
	Literatur . . . . .	10

# 1 Sekundärindexe

allgemein: Sekundärindex = Index, der einem Sekundärschlüsselwert eine Trefferliste zuordnet

“Trefferliste” im Kontext von XSLT: Menge von Knoten des Eingabe- baums

## 1.1 Anlegen eines Sekundärindexes

XSLT-Kommando `xsl:key`

```
<!-- Category: top-level-element -->
<xsl:key  name ='qname'
          match='pattern'
          use  ='expression' />
```

- Parameter `name`: Name des Sekundärindex; es können beliebig viele angelegt werden, die später durch ihren Namen identifiziert werden
- Parameter `match`: Typen der Knoten, die indexiert werden sollen (wie Parameter `match` in Transformationsregeln)
- Parameter `use`: Ausdruck, der zu einem Knoten dessen Sekundärschlüsselwert berechnet

Effekt der Ausführung eines `xsl:key`-Kommandos:

1. bestimme *alle Knoten im Eingabebaum*, die zu `match` passen
2. für jeden dieser Knoten: berechne den zugehörigen Sekundärschlüsselwert gemäß Parameter `use`
  - kann aus mehreren Datenwerten mit Textfunktionen wie `concat(..)`, `substring(..)` usw. konstruiert werden
  - Textfunktionen: s. XPath-Standard, 4.2 String Functions
3. bestimme zu jedem aufgetretenen Sekundärschlüsselwert die Treffer- liste, also die Liste der Knoten mit diesem Sekundärschlüsselwert

## 1.2 Benutzen eines Sekundärindexes

Abruf der Trefferliste für einen Sekundärschlüsselwert mit der Funktion:

```
key ( SName: string, SWert: object ) : node-set
```

- 1. Parameter: Bezeichner des Sekundärindexes
- 2. Parameter: Sekundärschlüsselwert
- Rückgabe: Trefferliste, also Liste von Referenzen auf Knoten des Eingabebaums

Funktion `key` hätte besser `getNodesForKeyValue` o.ä. geheißen

**Beispiel 1:** zeige alle Lehrveranstaltungen in einem bestimmten Semester an:

```
<xsl:key name =’LVproSemester’
          match=’ DURCHFUEHRUNG ’
          use   =’ @semester ’      />

<xsl:template match=’ / ’>
  <xsl:for-each
    select=’ key( “LVproSemester”, “2006s” ) ’ >
    <xsl:value-of select=’ @semester ’ />
    <xsl:value-of select=’ @dozentId ’ />
    <xsl:value-of select=’ .. / LEHRVERANSTALTUNGNAME ’ />
  </xsl:for-each>
</xsl:template>
```

## 2 Verbundbildung mit Index

**Beispiel 2:** wie oben (Verbundbildung mit Variablen)

```
<xsl:key name =’personendaten’
          match=’PERSON’
          use   =’@persId’      />

<xsl:template match=’ VERANTWORTLICHER ’ >
```

```

<VERANTWORTLICHER dozentId='{$dozentId}'>

  <xsl:for-each select='key( "personendaten", $dozentId )'>
    <xsl:value-of select='@nachname' />,
    <xsl:value-of select='@vornameInit' /> ...
  </xsl:for-each>

</VERANTWORTLICHER>
</xsl:template>

```

alternative Lösung mit einer Variablen statt `xsl:for-each`:

```

<xsl:key name ='personendaten'
  match='PERSON'
  use  ='@persId'      />

<xsl:template match=' VERANTWORTLICHER ' >

  <xsl:variable name="V"
    select='key( "personendaten", $dozentId )' />

  <VERANTWORTLICHER dozentId='{$dozentId}'>
    <xsl:value-of select=' $V / @nachname ' />,
    <xsl:value-of select=' $V / @vornameInit ' /> ...
  </VERANTWORTLICHER>

</xsl:template>

```

## 2.1 Verbundbildung als “Anreicherung” eines Elements

Typische Anforderung:

- vorhandenes (ggf. komplexes) Element enthält eine oder mehrere Referenzen auf andere Elemente (= Verbundpartner)
- Daten von den Verbundpartnern sollen zu diesem Element “hinzukopiert” werden (um alles lokal zu haben und ggf. in einer Pipeline weiterzuverarbeiten)
- das Element soll also komplett in die Ausgabe kopiert werden, ergänzt um zusätzliche Daten

Beispiel: Personendaten von **VERANTWORTLICHER** im vorigen Beispiel

**Lösungsschema** (X = Typ des Elements):

- a. für jedes Referenzattribut (bzw. einen entsprechenden Schlüsselwert) einen passenden Sekundärindex auf die Zielelemente anlegen
- b. generell identische Transformationsregel benutzen
- c. spezielle Transformationsregel für X, die folgendes kopiert:
  1. die Attribute von X
  2. die "hinzukopierten" Attribute von den Verbundpartnern
  3. die Kinder von X
  4. Kinder der Verbundpartner

```
<xsl:key name ='personendaten' .... />

<xsl:include href="identisch.xslt" />

<xsl:template match=' VERANTWORTLICHER ' >
  <xsl:copy>

    <!-- 1. lokale Attribute kopieren -->
    <xsl:apply-templates select=' @*' />

    <!-- 2. entfernte Attribute kopieren -->
    <xsl:apply-templates select='
      key( "personendaten", @dozentId ) / @*' />

    <!-- 3. lokale Kinder kopieren -->
    <xsl:apply-templates select=" node() " />
    ....
  </xsl:copy>
</xsl:template>
```

Erläuterungen:

- `<xsl:apply-templates select=' node() | @*' />` aus der identischen Transformation geht nicht als 1. Schritt, weil zuerst alle Attribute in der Ausgabe erzeugt werden müssen (vor den children)

- Alternativen zu `... / @*` in Schritt 2:
  - `... / @nameEinesAttributs`
  - `... / @* [ name(.)='fachgr' or name(.)='nachname' ]`  
Textfunktion `name()` liefert Namen (Typ) eines Element- oder Attributknotens

### 3 Gruppierung und Aggregation mit Index

Beispiele mit Zählung / Summierung:

Zahl der Module:

```
<xsl:value-of select=
    ' count(key("LVproSemester", "2006s")) ' />
```

Gesamtzahl der LP:

```
<xsl:value-of select=
    ' sum(key("LVproSemester", "2006s") / . . /
    LEISTUNGSPUNKTE / @anzahl) ' />
```

#### 3.1 Duplikateliminierung

Beispielaufgabe: zeige pro Semester die dort stattfindenden Lehrveranstaltungen

Problem:

- die Semesterkürzel treten in den Attributen `//DURCHFUEHRUNG/@semester` mehrfach auf
- ausgegeben werden soll *nur 1 Eintrag pro auftretendem Datenwert* (Semesterkürzel)
- die wesentliche Arbeit, insb. die Duplikateliminierung, wird im Prinzip beim Anlegen eines Sekundärindexes geleistet:

```
<xsl:key name ='semkrzl2semester'
    match='DURCHFUEHRUNG/@semester'
    use   ='.' />
```

Dieser SI enthält *pro Wert*, der in den Attributen @semester auftritt, *einen Eintrag*; die zug. Trefferliste enthält Referenzen auf alle Attribut-Knoten in der Eingabe, wo dieser Wert vorkommt.

- Problem: es gibt keinen Iterator für SI! (der über alle Trefferlisten in dem SI iteriert)

Ersatzkonstruktion für einen Iterator:

```
<xsl:variable name='alleVerschiedenenSemkrzl'
  select=' //DURCHFUEHRUNG/@semester
  [ generate-id( . ) =
    generate-id( key( "semkrzl2semester", . ) [1] )
  ]' />
```

- alle @semester-Knoten werden erneut durchsucht
- selektiert werden die Knoten, die in der zug. Trefferliste (`key( 'semkrzl2semester', . )`) auf Platz eins ([1]) stehen
- Um *Referenzen* auf Konten in der Eingabe *vergleichen* zu können, müssen die Referenzen erst in einen String umgewandelt werden; hierzu: `generate-id()`

Im Endeffekt enthält die Variable `alleVerschiedenenSemkrzl` für jeden Wert, der in der Eingabe vorkommt, genau eine Referenz auf einen @semester-Knoten in der Eingabe, in dem dieser Wert steht.

Testfrage: warum funktioniert die folgende Lösung ohne `generate-id()` nicht?

```
<xsl:variable name='alleVerschiedenenSemkrzl'
  select=' //DURCHFUEHRUNG/@semester
  [ . =
    key( "semkrzl2semester", . ) [1]
  ]' />
```

Antwort: Diese Lösung eliminiert die Duplikate nicht!

In dem Gleichheitsvergleich werden hier *die textuellen Werte der Knoten verglichen*, und nicht die Referenzen auf die Knoten im Eingaben.

Die textuellen Werte der Knoten sind bei allen Einträgen in der Tref-ferliste definitionsgemäß gleich!

Nutzung der “eindeutige-Werte-Variablen”:

1. Ausgabe der Anzahl der verschiedenen Werte:

```
<xsl:value-of select='count($alleVerschiedenenSemkrzl)' />
```

2. Als Iterator-Menge in for-each-Anweisungen, z.B. pro Semester eine Liste aller Lehrveranstaltungen in diesem Semester ausgeben:

```
<xsl:for-each select='$alleVerschiedenenSemkrzl' >
  <xsl:sort select='.' />
  ....
  <!-- zug. Gruppe von Knoten zum aktuellen Wert -->
  <xsl:variable name='aktuelleGruppe'
    select=' key( "semkrzl2semester" , . ) ' />
  ....
```

### 3.2 Bewertung von Indexen

- wesentlich effizienter als viele direkte Abfragen, wenn der gleiche Datenbestand wiederholt durchsucht werden muß
- günstig für Verbundbildung oder Gruppierung / Aggregation
- unverständliche Namen von Kommando / Funktion

## 4 Softwaretechnische Beurteilung von XSLT

### Vorteile von XSLT:

- bei sehr einfachen Web-Applikationen: nur 1 Sprache  
Indiz, daß pures XSLT sinnvoll ist: man kommt mit wenigen “kleinen” Transformationen (50 - 200 Zeilen) aus
- grundlegende Operatoren (Selektionen, Projektion, Verbund) schematisch mit Standard-Mustern sicher implementierbar

## Nachteile von XSLT:

- Lesbarkeit / linguistische Qualität der XSLT-Programme: desaströs, Fehlerquelle erster Güte
  - ... man könnte auch Java-Programme als Syntaxbaum in XML codiert darstellen!
- kein vernünftiges Modulkonzept
- ungeeignet für komplexere Algorithmen / Applikationen
  - beschränken auf reine Abfragezwecke
  - sinnvolle Trennung zwischen Datenextraktion – Fachlogik (in richtiger Programmiersprache!) anstreben

## Literatur

[XPAT] Kelter, U.: Lehrmodul “XPATH”; 2009

[XSLT] Kelter, U.: Lehrmodul “XSLT, Teil 1 (Stichworte)”; 2009

[XSLT2] Kelter, U.: Lehrmodul “XSLT, Teil 2 (Stichworte)”; 2009

[XSLT3] Kelter, U.: Lehrmodul “XSLT, Teil 3”; 2009