Querysprachen für XML

Zusätzliche Quellen:
http://www.w3.org/TR/xquery/
http://www.w3schools.com/xquery/

XML-QL

XML-QL ist eine Querysprache für XML mit folgenden Eigenschaften:

- Kombiniert die XML-Syntax mit QL-Techniken.
- Nutzt Pfadausdrücke und Patterns, um Daten aus den XML-Eingabedaten zu extrahieren.
- Daten werden an Variablen gebunden.
- Die XML-Ausgabedaten werden nach Templates konstruiert.
- Patterns und Templates nutzen die XML-Syntax.

Semistrukturierte Datenmodell und XML Datenmodell

- XML-QL besitzt ein semi-strukturiertes Datenmodell.
- Daten werden als kantenbeschriftete Graphen repräsentiert, während das XML-Datenmodell durch knotenbeschriftete Graphen repräsentiert wird.
- Übersetzungen zwischen beiden Datenmodelle sind möglich.
- XML-QL basiert auf einer **`where/construct`** Syntax anstatt der **`select/from/where`** Syntax.
- Die **`construct`**-Klausel entspricht der **`select`**-Klausel und die **`where`**-Klausel der herkömmlichen **`from`**- und **`where`**-Klausel.

Beispiel

```
<answer>
   where <book>
           <publisher><name> Morgan Kaufmann </></>
           <title> $T </>
           <author> $A </>
         </> in "www.a.b.c/bib.xml"
   construct <result>
                <booktitle> $T </booktitle>
              where <price> $P </price> in $B
              construct <bookprice> $P </bookprice>
              </>
</answer>
```

XQuery

XQuery ist eine Querysprache für XML:

- Basiert unter anderem auf XML-QL und OQL.
- Definition: Eine Sprache ist *abgeschlossen* in Bezug auf ein Datenmodell, wenn das Ergebnis jeden Ausdrucks dieser Sprache wieder im Datenmodell liegt.
- XQuery ist abgeschlossen in Bezug auf das Zugehörige Datenmodell.

Datenmodell

- Das Datenmodell unterstütz XML Schema. XML Schema definiert Eigenschaften wie die Struktur und einfachen Datentypen.
- Repräsentiert Sammlungen von Dokumenten und komplexen Werten.
- Unterstützt typisierte atomare Werte.
- Unterstützt geordnete heterogene Sequenzen.
- Das Datenmodell spezifiziert, welche Informationen in einem Dokument zugreifbar sind.
- Es spezifiziert nicht das Programmierspracheninterface oder die Bindungen die genutzt werden um Daten darzustellen oder auf Daten zuzugreifen.

Formale Semantik

- XQuery ist eine funktionale Sprache.
- XQuery basiert eher auf Ausdrücken als auf Anweisungen.
- Alle Konstrukte der Sprache (außer dem XQuery Anfrageprolog) ist ein Ausdruck. Diese Ausdrücke können willkürlich kombiniert werden.
- Das Ergebnis eines Ausdrucks kann als Eingabe für den nächsten Ausdruck dienen, solange der Typ des Ergebnisses mit dem Typ der Eingabe übereinstimmt.
- Eine weiter Eigenschaft einer funktionalen Sprache ist, dass Variablen als Werte weitergegeben werden. Werte können nicht von Seiteneffekten betroffen werden.

Formale Semantik

- XQuery ist eine typisierte Sprache.
- Das Typsystem basiert auf XML-Schema (eine Erweiterung von DTD).
- Es können Typen aus XML-Schemata, die die Eingabe- und die Ausgabedokumente beschreiben, importiert werden und anschließend können Operationen auf diesen Typen ausgeführt werden.
- Das Typsystem von XQuery unterstützt (Ausschnitt):
  - Globale und lokale Element- und Attributdeklarationen.
  - Komplexe und einfache Typdefinitionen.
  - Benannte und anonyme Typen.
  - Listen und Vereinigungen.
  - Wildcards.

Statische Typisierung

- XQuery unterstützt die statische Typanalyse.
- Die statische Typanalyse leitet den Ausgabetyp eines Ausdrucks von den Typen seiner Eingaben ab.
- Statische Typisierung wird genutzt um Typfehler zu erkennen und dient als Basis für Optimierungen.

Ausdrücke

- Ausdrücke sind die Basis von XQuery.
- Ausdrücke sind Zeichenketten von Unicodezeichen.
- XQuery kennt verschiedene Typen von Ausdrücken, die aus Schlüsselwörtern, Symbolen und Operationen konstruiert werden.
- Die Operationen eines Ausdrucks sind wiederum Ausdrücke.
- In XQuery haben eingebetete Ausdrücke ihre volle Aussagekraft.

Ausdrücke

- Der statische Kontext:
  - eines Ausdrucks sind die Informationen, die während einer statischen Analyse des Ausdrucks, vor dessen Auswertung, verfügbar sind.
  - Diese Informationen können genutzt werden um statische Fehler zu finden.
- Der dynamische Kontext:
  - eines Ausdrucks, sind die Informationen, die während der der Auswertung des Ausdrucks, verfügbar sind.
  - Diese Informationen können genutzt werden um dynamische Fehler zu finden.

XML-Namensräume

- Ein XML-Namensraum ist eine Sammlung von Namen, identifiziert durch eine URI-Referenz, die in einem XML-Dokument als Elementtypen und Attributnamen verwendet werden.
- XML-Namensräume unterscheiden sich von herkömmlichen Namensräumen dadurch, dass sie keine Menge (im mathematischen Sinn) darstellen sondern eine interne Struktur besitzen.

```
<x xmlns:edi='http://ecommerce.org/schema'>
  <!-- the "edi" prefix is bound to
       http://ecommerce.org/schema
       for the "x" element and contents -->
</x>
```

Quelle: http://www.w3.org/TR/1999/REC-xml-names-19990114/

Konzepte von XQuery

- Dokumentenordnung
  - Die Dokumentenordnung legt die Ordnung aller Knoten, die während einer Query zugreifbar sind, fest. Sie ist total und stabil.
  - Informale Definition: Die Dokumentenordnung ist die Ordnung in der die Knoten in der XML-Anordnung eines Dokumentes vorkommen.
- Atomisierung
  - Die Semantik mancher Operatoren benötigt den Prozess der Atomisierung.
  - Atomisierung wird dann auf einen Wert angewandt, wenn der Wert in einem Kontext genutzt wird, indem eine Sequenz von atomaren Werten gebraucht wird.

Konzepte von XQuery

- Eingabequellen
  - XQuery hat eine Reihe von Funktionen die Zugriff auf Eingabedaten gewährt.
  - Mit Hilfe dieser Funktionen können Ausdrücke auf Dokumente verweisen.
- URI-Literale
  - Ein URI-Literal verweist innerhalb einer XQuery auf eine statische gültige absolute URI.
  - Zum Beispiel werden über URI-Literale Namensräume definiert die statisch bekannt sein müssen.

XQuery Syntaxregeln (Ausschnitt)

- XQuery ist case-sensitive.
- Elemente, Attribute und Variablen müssen gültige XML Namen sein.
- Eine Variable wird mit einem $ gefolgt vom Namen definiert. Beispiel:
  `$titel`
- Kommentare werden mit (: und :) eingefasst. Beispiel:
  `(: Ein Kommentar :)`

Bedingte Ausdrücke

- In XQuery sind „If-Then-Else“-Ausdrücke möglich:

```
for $x in doc("books.xml")/bookstore/book
return if ($x/@category="CHILDREN")
       then <child>{data($x/title)}</child>
       else <adult>{data($x/title)}</adult>
```

- **`else`** ist notwendig kann aber leer sein: **`else ()`**

Vergleichen

- XQuery kennt zwei Wege Werte zu vergleichen.
  1. Allgemeine Vergleiche: **`=, !=, <, <=, >, >=`**
  2. Wertvergleiche: **`eq, ne, lt, le, gt, ge`**
- **`$bookstore//book/@q > 10`**
  Der Ausdruck liefert wahr zurück, wenn alle Attribute **`q`** einen Wert größer als 10 haben.
- **`$bookstore//book/@q gt 10`**
  Dieser Ausdruck liefert wahr zurück, wenn der Ausdruck nur ein Attribut **`q`** zurückliefert und dessen Wert größer als 10 ist. Wenn mehr als ein **`q`** zurückgegeben wird, tritt ein Fehler auf.

Selektion und Filterung

- In XQuery existieren verschiedene Elemente zum Selektieren und Filtern:
  - **`for`**
  - **`let`**
  - **`where`**
  - **`order by`**
  - **`return`**

`for`-Klausel

- Die **`for`**-Klausel bindet eine Variable zu jedem Objekt, das der **`in`**-Ausdruck zurückliefert. Die **`for`**-Klausel entspricht einer Iteration.

```
for $x in (1 to 5)
return <test>{$x}</test>
```


`let`-Klausel

- Die **`let`**-Klausel ermöglicht variable Zuordnungen und verhindert die Wiederholungen des selben Ausdrucks. Die **`let`**-Klausel stellt keine Iteration dar.

```
let $x := (1 to 5)
return <test>{$x}</test>
```

ergibt:

```
<test>1 2 3 4 5</test>
```

`where`-Klausel

- Die **`where`**-Klausel spezifiziert ein oder mehr Kriterien für das Ergebnis**.**

```
where $x/price>30 and
      $x/price<100
```


`order by`-Klausel

- Die **`order by`**-Klausel sortiert die Reihenfolge der Ergebnisse.

```
for $x in doc("bsp.xml")/regal
order by $x/@category, $x/titel
return $x/titel
```


`retrun`-Klausel

- Die **`return`**-Klausel gibt an was zurückgegeben wird.

```
for $x in doc(„bsp.xml")/regal
return $x/titel
```


Funktionen

- XQuery verfügt über Funktionen zur Bearbeitung von Zeichenketten, Zahlen und Booleanwerten, zum Vergleich von Datums- und Zeitangaben, usw..
- Außerdem können Funktionen vom Benutzer definiert werden.
- Funktionen befinden sich im Funktionen-namensraum „fn“. Jedoch kann das Präfix fn: bei Funktionsaufrufen weggelassen werden.

Funktionssignaturen

- Jede Funktion ist durch eine spezifische Signatur, eine Beschreibung des Rückgabetyps und aller Parameter und ihrer Semantik, definiert.
- Die Funktionssignatur hat folgende Form:

```
fn:function-name($parameter-name as parameter-type,
                 ...) as return-type
```


Beispiele

- Ein Funktionsaufruf kann überall da vorkommen, wo auch ein Ausdruck vorkommen kann.
- Zum Beispiel in einen Element:
  `<name>{uppercase($titel)}</name>`
- Zum Beispiel als Prädikat eines Pfadausdrucks:
  `doc(„bsp.xml")/regal/buch[substring(titel,1,7)=,Data on']`
- Oder in einer Klausel:
  `let $name := (substring($titel,1,4))`

Benutzerdefinierte Funktionen

- Benutzerdefinierte Funktionen können innerhalb einer Anfrage oder in einer separaten Bibliothek definiert werden.
- Anmerkungen:
  - Der Name der Funktion muss einen Präfix haben.
  - Die Datentypen der Parameter sind in der Regel die selben Datentypen wie in XML-Schema.

Beispiel

```
declare function local:minPrice(
  $price as xs:decimal?,
  $discount as xs:decimal?)
  AS xs:decimal?
{
  let $disc := ($price * $discount) div 100
  return ($price - $disc)
};

(:Aufruf:)
<minPrice>
  {local:minPrice($book/price,$book/discount)}
</minPrice>
```

XSL

XSL (extensible Stylesheet Language) hat folgende Eigenschaften:

- XSL erlaubt, Transformationen von XML nach HTML zu beschreiben. (Die Beschriebung der Präsentation eines XML-Dokuments.)
- Sie wird in Datenapplikationen eingesetzt.
- Sie ist nicht relational vollständig (relationally complete).
- Sie kann keine Joins.
- Die möglichen Transformationen sind beschränkt.

XSL Datenmodell

- Das XSL-Datenmodell entspricht genau dem XML-Datenmodell.
- Alle XML-Konstrukte werden von XSL beherrscht.
- Das Datenmodell ist ein geordneter Baum.
- Referenzen und Links müssen separat behandelt werden.

XSL-Programme

- Ein XSL-Programm ist eine Menge von Template Regeln (template rules).
- Jede Regel besteht aus einem Pattern und einem Template.
- XSL startet mit einem Wurzelelement und versucht ein Pattern an diesem Knoten anzuwenden. Wenn es passt, wird das zugehörige Template ausgeführt.
- Das Template weist XSL an, ein XML Ergebnis zu produzieren und das Template rekursiv an dem Kindknoten anzuwenden.
- ⇒ XSL Programme sind wie rekursive Funktionen.

Beispiel

XML Dokument:

```
<bib> <book> <title>  t1 </title>
             <author> a1 </author>
             <author> a2 </author>
      </book>
      <paper><title>  t2 </title>
             <author> a3 </author>
             <author> a4 </author>
      </paper>
      <book> <title>  t3 </title>
             <author> a5 </author>
             <author> a6 </author>
             <author> a7 </author>
      </book>
</bib>
```

Beispiel

XSL Programm:

```
<xsl:template>
    <xsl:apply-templates/>
</xsl:template>
<xsl:template match="/bib/*/title">
    <result>
        <xsl:value-of/>
    </result>
</xsl:template>
```

Das Ergebnis:

```
<result> t1 </result>
<result> t2 </result>
<result> t3 </result>
```


XSL-Datenmodell

- Es besteht aus einem Baum mit einem einzigen Wurzelknoten, der das ganze Dokument kennzeichnet.
- Dieser Wurzelknoten differenziert sich vom obersten Element des XML-Dokuments.
- Die Wurzel wird nur durch „/“ gematcht und hat ein einziges Kind, das oberste Element vom XML Dokument.
- In XSL sind Kommentare und „Processing Instructions“ Teil des Datenmodells.

Pattern und Variablen

- Pattern sind Pfadausdrücke.
- Um nicht-lineare Pattern definieren zu können, werden Abfragekriterien verwendet, die mit `[...]` eingeschlossen werden. Beispiel:
  `bib/paper[year and publisher/name and @language]`
- XSL benutzt keine Variablen.
  - Vorteil: kurze und knappe Programme
  - Nachteil: Einschränkung der Ausdruckskraft
    ⇒ z.B. Joins kann XSL nicht ausdrücken.

XSL-Templates

- Das Template enthält eine Mischung von zu generierendem XML-Text und XSL-Befehlen.

```
<xsl:template match="pattern">
    template
</xsl:template>
```

Namensraum der XSL-Elemente

- XSL Befehle werden dadurch identifiziert, dass ihre Elementnamen mit **`xsl:`** anfangen.
- Sie gehören zum Namensraum von XSL. Diese Befehle referenzieren immer zum aktuellen Knoten.
- Beispiele:
  - **`<xsl:value-of>`** wertet den Wert des aktuellen Knotens aus. Wenn der Knoten nicht atomar ist, ist der Wert die String Konkatenation der Werte der Kinder.
  - **`<xsl:element name="...">`** kreiert ein neues Element mit Namen ...

Beispiel (XSL-Elemente)

Diese Beispiele sind zueinander äquivalent:

```
<xsl:template match="A">
    <xsl:element name="B">
        <xsl:value-of/>
    </xsl:element>
</xsl:template>

<xsl:template match="A">
    <B>
        <xsl:value-of/>
    </B>
</xsl:template>
```


Beispiel (XSL-Elemente)

```
<xsl:template match="*">
    <xsl:element name="name()">
        <xsl:value-of/>
    </xsl:element>
</xsl:template>
```