

XQuery

Literatur

- [1] "XQuery"; Wolfgang Lehner, Harald Schöning;
dpunkt.verlag, 2004; ISBN 3-89864-266-6.
- [2] "XQuery, The XML Query Language";
Michael Brundage; Addison-Wesley, 2004;
ISBN 0-321-16581-0.
- [3] "XQuery from the Experts "; Don Chamberlain;
Addison-Wesley, 2004; ISBN 0-321-18060-7.
- [4] "XML & Datenbanken";
Meike Klettke, Holger Meyer; dpunkt.verlag, Dezember
2002; ISBN 3-89864-148-1.
- [5] "XML in a Nutshell";
Elliotte Rusty Harold, W. Scott Means; O'Reilly, June 2002;
ISBN 0-596-00292-0.

I
XPath

XPath, Bedeutung aus DB-Sicht

- Aus klassischer Sicht ist folgende SQL-Tabelle nicht korrekt, da sie ein mehrwertiges Attribut enthält.

isbn	titel	preis	beschreibung
1-23-444-9	Learning XML	23.50 €	<desc> <de>sehr gut</de> <fr>tres bien</fr> <en>very good</en> </desc>
56-981-67	XML Tricks	56 €	<desc> <de>unbrauchbar</de> <fr>inutilisable</fr> <en>useless</en> </desc>

- In SQL müsste eine Dekomposition in zwei Tabellen vorgenommen werden.

XPath, Bedeutung aus DB-Sicht

- Viele Datenbanken stellen heute den Datentyp XML zur Verfügung und entsprechende Ergänzungen von SQL, mit welchen gezielt auf Substrukten zugegriffen werden kann.
- Beispiel einer Abfrage in Oracle SQL, um Bücher mit englischer Beschreibung auszugeben, die billiger als 50 € sind:

```
select isbn, titel, preis,  
beschreibung.extract( '/desc/en' ).getStringVal()  
from Buch  
where preis < 50
```

→ Beilage: Oracle XML-Beispiele

Analog zum Abfragen existieren Konstrukte für das Einfügen und Ändern von Substrukturen in einem XML-Datenfeld.

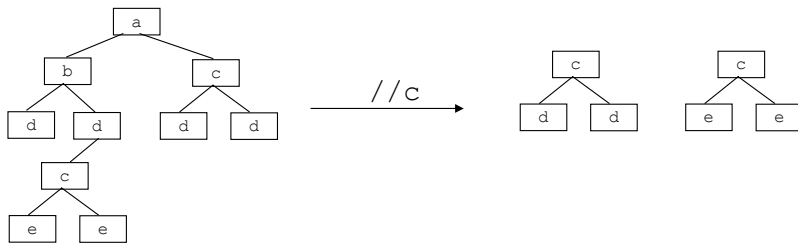
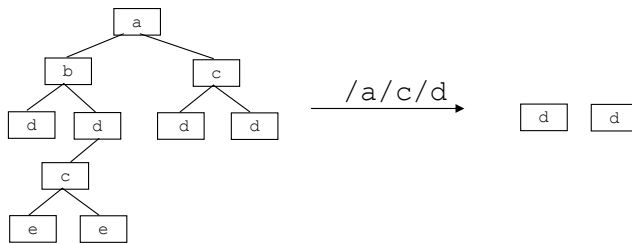
XPath, Übersicht

- XPath ist eine Sprache zur Lokalisierung (Adressierung) von Knoten in einem XML-Dokument.
- Das Resultat einer XPath-Abfrage ist eine Sequenz von Knoten, resp. der mit diesen Knoten angesprochenen Teilbäume.
- Die Lokalisierung (Adressierung) von Knoten erinnert stark an die Adressierung von Dateien in einem Filesystem.

XPath wirkt genau genommen nicht nur auf XML-Dokumente, sondern auf eine beliebige Sequenz von XML-Knoten (Dokumente, Elemente, Attribute, Text, Processing-Instructions, Kommentare) oder atomaren Werten. Häufig besteht diese Sequenz jedoch nur aus einem einzigen Dokumentknoten.

XPath kann im Gegensatz zu XQuery keine *neuen* XML-Strukturen erzeugen, ist aber die Grundlage von XQuery zur Entnahme von Knoten resp. Teilbäumen und zur Konstruktion neuer XML-Strukturen.

XPath, Pfadausdrücke



XPath, Ausgangspunkte

- Der Ausgangspunkt für eine XPath-Abfrage ist häufig ein XML-Dokument:

`doc( "catalog-generic.xml" ) /Xpath-Ausdruck`

- In einem Programm oder in einem XSL Style-Sheet kann der Ausgangspunkt eine geöffnete XML-Datei sein. Dann beginnt die XPath-Abfrage einfach mit einem Schrägstrich:

`/Xpath-Ausdruck`

- In XQuery ist der Ausgangspunkt häufig eine Variable, welche einen oder mehrere XML-Knoten enthält:

`$x /Xpath-Ausdruck`

XPath, Lokalisierung nach Name

Lokalisierung von Elementen nach Name

- 1) `/reservations`
- 2) `/reservations/reservation/book`
- 3) `//book`
- 4) `../book`
- 5) `//book/(isbn|title)`
- 6) `//book/*`
- 7) `//*`

Lokalisierung von Attributen nach Name

- 1) `//reservation/@resid`
- 2) `//reservation/@*`
- 3) `//@*`

Lokalisierung von Elementen nach Name

- 1) Der erste Schrägstrich repräsentiert den Ausgangspunkt, also beispielsweise das XML-Dokument als Ganzes (nicht den Wurzelknoten!). Ausgehend davon wird nach Elementen mit dem Namen 'reservations' gesucht.
- 2) Es wird in drei Schritten ab dem Ausgangspunkt die Menge aller Elemente mit dem Namen 'reservations', ausgehend von diesen die Elemente mit dem Namen 'reservation', ausgehend von diesen die Elemente mit dem Namen 'book' gesucht.
- 3) Es wird ab dem Ausgangspunkt nach book-Elementen in *beliebiger* Tiefe gesucht.
- 4) Dies ist ein *relativer* Pfadausdruck. Ein relativer Ausdruck macht nur Sinn, wenn ein *Kontextknoten* definiert ist. Dies ist beispielsweise bei der Angabe von Bedingungen der Fall. Im folgenden Beispiel werden alle Buchtitel aus dem Jahr 2001 gesucht: `//book/title[../year='2001']`
- 5) Es wird ab dem Ausgangspunkt nach isbn-Elementen oder title-Elementen gesucht, die unmittelbare Kinder eines book-Elementes sind.
- 6) Es werden alle unmittelbaren Kindelemente von book-Elementen zurückgeliefert.
- 7) Es werden alle Knoten in beliebiger Tiefe zurückgeliefert, inklusive dem Ausgangspunkt selbst.

Lokalisierung von Attributen nach Name

- 1) Es werden resid-Attribute gesucht, die zu einem reservation-Element gehören.
- 2) Es werden alle Attribute gesucht, die zu einem reservation-Element gehören.
- 3) Es werden alle Attribute von Elementen in beliebiger Tiefe gesucht.

XPath, Lokalisierung nach Knotentyp

- 1) `//book/processing-instruction()`
- 2) `//book/comment()`
- 3) `//book/attribute()`
- 4) `//book/*/text()`
- 5) `//book/node()`

- 1) Es werden alle Processing Instructions gesucht, die unmittelbare Kinder von book-Elementen sind.
- 2) Es werden alle Kommentare gesucht, die unmittelbare Kinder von book-Elementen sind.
- 3) Es werden alle Attribute gesucht, die unmittelbare Kinder von book-Elementen sind. Diese Abfrage ist identisch mit `//book/@*`
- 4) Es werden alle textuellen Inhalte von Elementen unterhalb von book-Elementen gesucht.
- 5) Es werden alle Arten von Knoten unterhalb von book-Elementen gesucht. Es ist zu beachten, dass die Abfrage `//book/*` nur die *Elemente* unterhalb von book-Elementen liefert. Unterhalb von book könnten aber auch noch Kommentare, Processing Instructions usw. vorkommen.

XPath, Prädikate

- In XPath können, wie in XQuery, Prädikate (Bedingungen) eingesetzt werden.

- 1) `/catalog/book[title = 'Learning XML']`
- 2) `/catalog/book/title[. = 'Learning XML']`
- 3) `/catalog/book[./title = 'Learning XML']`
- 4) `//book[pages < 600]/title`
- 5) `//reservation[book/isbn = '0596000464']/person`
- 6) `//reservation/person[../book/isbn = '0596000464']`

→ Beilage: XPath Ausdruecke

→ Beilage: Funktionen und Operatoren

In vielen Anwendungsfällen wird eine Bedingung in einem XPath-Ausdruck *oder* in der where-Klausel eines XQuery-Ausdruckes eingesetzt werden können. Welche Variante gewählt wird, bleibt dem Entwickler überlassen. Eventuell ergeben sich Performance-Unterschiede.

- 1) innerhalb der Bedingung gilt jeweils der Knoten bei dem die Bedingung steht, als *Kontextknoten*. Das heisst, das `title`-Element wird als unterhalb zum `book`-Element angesehen. Die Bedingung könnte auch ausführlicher als `[./title = 'Learning XML']` geschrieben werden.
- 2) Der Punkt steht stellvertretend für den Kontextknoten, das heisst in diesem Fall das `title`-Element selbst.
- 3) Der Ausdruck `./title` heisst: Suche eine `title`-Element in beliebiger Tiefe, ausgehend vom aktuellen Kontextknoten.
- 4) Die Bedingung muss keineswegs im letzten Schritt, sondern kann an einer oder mehreren Stellen im XPath-Ausdruck stehen.
- 5) Innherhalb der Bedingung kann eine Navigation nach oben oder unten im XML-Dokument stattfinden.
- 6) Diese Abfrage ist resultatmässig identisch zur Abfrage 5)

XPath, Prädikate ff

```
7)    //reservation[last()]
8)    //reservation[position()=1]
9)    //book[contains(title, 'Learning')]
10)   //book[title='Learning XML'
           or title='XML Bible']
```

→ Beilage: XPath Ausdruecke

→ Beilage: Funktionen und Operatoren

- 7) Die Bedingung kann eine Positionsangabe beinhalten. In diesem Beispiel wird für jeden Pfad, der zu einer Menge von Reservationselementen führt, das erste Reservationselement herausgezogen.
- 8) siehe 7)
- 9) Die contains()-Funktions hat als ersten Parameter ein title-Element, dessen Kontextknoten das book-Element ist. und als zweiten Parameter einen String. Viele Funktionen in XPath übernehmen den Kontextknoten als impliziten Parameter.
- 10) Bedingungen können or, and oder not Operatoren enthalten.

XPath, Adressierungsachsen

- Bisher wurde nur die abgekürzte Schreibweise von XPath verwendet. Es gibt eine systematischere, mächtigere, aber etwas umständlichere Schreibweise. Ein wesentliches Charakteristikum darin sind Adressierungsachsen:

```
/child::catalog/child::book  
/child::catalog/descendant::book  
/child::catalog/child::book[title='XML']/preceding-sibling::*
```

→ Exakte Grammatik unter <http://www.w3.org/TR/xpath20/>

In der abgekürzten Schreibweise entspricht der Schrägstrich der Achsenangabe child::. In der systematischen Schreibweise wird der Schrägstrich zum reinen Trennzeichen für zwei Lokalisierungsschritte. Folgende Adressierungsachsen sind definiert:

```
self::  
child::  
parent::  
descendant::  
descendant-or-self::  
ancestor::  
ancestor-or-self::  
preceding::  
preceding-sibling::  
following::  
following-sibling::  
attribute::
```

Der Wildcard * entspricht damit dem Ausdruck child::*

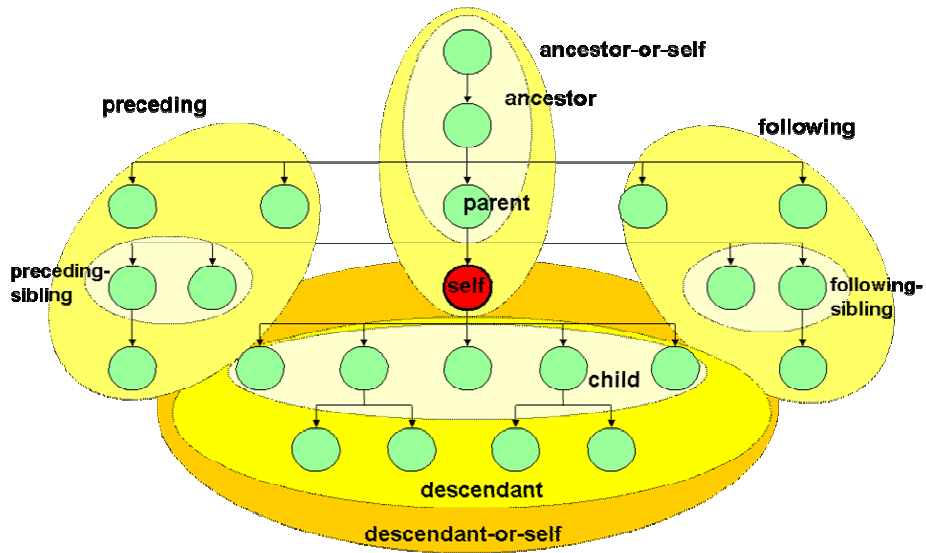
Der Wildcard //* entspricht damit dem Ausdruck descendant-or-self::*

Der Wildcard @* entspricht damit dem Ausdruck attribute::*

Der Punkt . entspricht damit dem Ausdruck self::*

Der Zweifachpunkt .. entspricht damit dem Ausdruck parent::*

XPath, Adressierungsachsen



II XQuery Übersicht

Die Grenzen meiner Sprache bedeuten die Grenzen meiner Welt
[Ludwig Wittgenstein, Philosoph, 1889 – 1951]

Welche Probleme löst XQuery?

- Abfragen in Datenbeständen mit komplexer Struktur
 - Textdokumente (z.B. OpenOffice!)
 - Mehrsprachige Texte, Fragebogen, Umfragen
- Dokumente an bestimmten Stellen mit Aktualdaten ergänzen ("Serienbriefe"-Konzept).
- Im Gegensatz zu XPath können mit XQuery *neue* XML-Fragmente oder XML-Dokumente aus mehreren Quellen zusammengesetzt werden.

Merkmale von XQuery

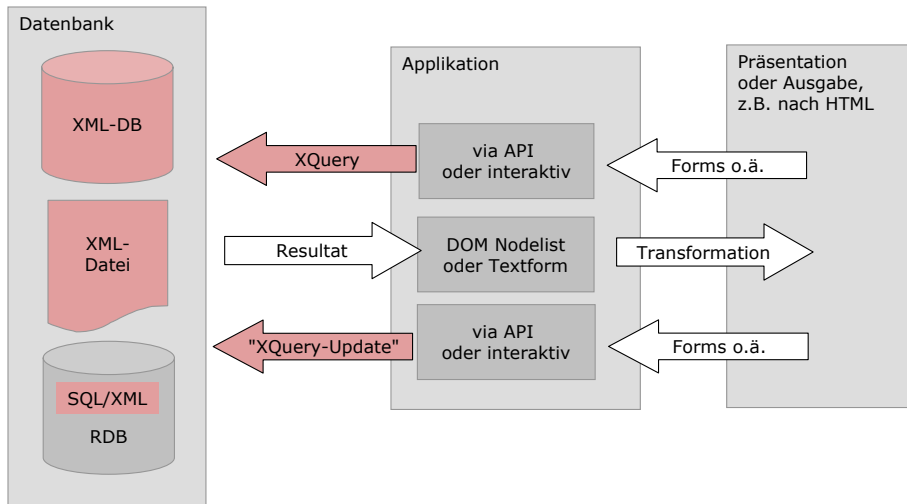
- Standardisierung durch W3C, Draft-Status seit 1999. Der Übergang zur Recommendation ist klar beabsichtigt.
- Einige *wichtige DB-Hersteller* bieten XQuery-Unterstützung an. Viele Implementation auf Basis XML-Dateien vorhanden.
- XQuery überlappt vom Anwendungsgebiet her mit XSLT, SQL, und XPath. XPath ist ausserdem ein Teil der Spezifikation von XQuery.
- XQuery kann sowohl mit untypisierten und typisierten Daten umgehen.

Umgang mit untypisierten Daten heisst beispielsweise, dass folgender Vergleich als false ausgewertet wird, weil untypisiert in den meisten Fällen gleichbedeutend mit einer Behandlung als String ist:

```
<A>1.0</A> eq <A>1</A>
```

Ist A jedoch ein typisiertes Element, weil es mit einem entsprechenden XML Schema hinterlegt ist, das A als Zahl und nicht als String ausweist, so ergibt obiger Vergleich true.

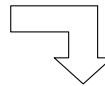
Wie wird XQuery eingesetzt?



Ein Java-API für XQuery von Sun ist in Vorbereitung unter dem Namen XQJ. Dieses ist aber erst im Status einer 'Early Draft Review'. XQuery ist eine reine Abfragesprache. Änderungen von Knoten, die via XQuery an eine Applikation ausgeliefert werden, sind nicht möglich. Einige Hersteller bieten Erweiterungen an, mit denen Updates durchgeführt werden können. Der Standardisierungsprozess beim W3C ist in dieser Hinsicht aber noch in den Anfängen. Häufig wird eine XPath-Schnittstelle angeboten, über die DOM-Dokumente oder DOM-Nodelists abgefragt werden können. Diese sind dann über die DOM-Funktionen veränderbar.

Ein XQuery-Beispiel, 1

```
<result> {  
  for $book in doc( "catalog-generic.xml" )//book  
  let $item := ( $book/title, $book/isbn )  
  where $book/title = 'XML in a Nutshell'  
  order by $item/title  
  return $item  
} </result>
```



```
<result>  
  <title>XML in a Nutshell</title>  
  <isbn>0596000588</isbn>  
</result>
```

Bemerkungen zu diesem Beispiel:

- Die Anfangsbuchstaben der Schlüsselwörter `for`, `let`, `where`, `order` und `return` ergeben FLWOR. Eine XQuery-Abfrage heisst daher oft auch "Flower"-Ausdruck.
 - Ein XQuery-Ausdruck ist in geschweifte Klammern `{ }` eingebettet. Um den Ausdruck herum kann reguläres XML stehen.
 - Die Funktion `doc( uri )` dient als Ausgangspunkt in die Datenbank oder ein XML-Dokument. Der `uri`-Parameter kann auf eine lokale Datei oder eine http-Lokation zeigen, welche ihrerseits wieder ein Dokument oder einen Datenbankzugriff referenziert. Die Details des `uri`-Parameters dürfen produktspezifisch sein.
 - Meistens wird auf den Ausgangspunkt der Abfrage ein XPath-Ausdruck angewendet, der eine Sequenz von Knoten erzeugt. Diese Knotensequenz wird dann innerhalb der `for`-Anweisung schrittweise abgearbeitet.
 - Die Schlüsselwörter `for`, `where`, `order by` und `return` entsprechen grob den Schlüsselwörtern `from`, `where`, `order by`, `select` in SQL. Das Schlüsselwort `let` erlaubt eine Variablen-Deklaration resp. Zuweisung.
 - Das Schlüsselwort `for` zusammen mit `where` definiert eine Sequenz von `book`-Elementen, auf die in `order by` Bezug genommen wird. Das Schlüsselwort `return` bezieht sich auf jedes einzelne Element, der durch das `for-where-order by`-Konstrukt definierte Sequenz.
-
- **rot:** Schlüsselworte von XQuery
 - **grün:** XPath-Ausdrücke

Ein XQuery-Beispiel, 2

Das folgende Beispiel ist eine *Änderungs*-Anfrage:

```
update  
let $doc := doc( "http://www.sws.bfh.ch/doc.xml" )  
for $book in $doc//book  
where $book/title = 'XML in a Nutshell'  
do replace $book/subtitle/text() with "2nd edition"
```

Eine standardisierte Sprache resp. ein XQuery-Zusatz für *Änderungs*-Anfragen steckt noch in den Anfängen. Aber: es gibt gute Produkte mit eigener Syntax.

Leider fehlt momentan noch ein *Standard*, um Daten über XQuery ändern zu können, analog zum insert-, update- oder delete-Befehl in SQL. Nichtsdestoweniger gibt es Produkte, die solche zusätzlichen Befehle kennen, wie das obige Beispiel mit Tamino, der XML-Datenbank von Software AG, zeigt.

Datenquellen für XQuery

- Externe Ausgangsdaten für eine XQuery-Abfrage werden von folgenden Methoden geliefert:

`doc( uri )`

liefert ein wohlgeformtes, unter Umständen validiertes XML-Dokument zurück. uri ist nicht genau spezifiziert, in der Regel ist es ein http- oder ein file-URL. Der Rückgabewert von doc() ist ein Dokumentknoten.

`collection( uri )`

liefert eine Sequenz von Knoten zurück. uri kann irgendeine Form einer Ressourcen-Adressierung sein. Die Implementationsart für collection() ist vorgeschrieben.

XQuery und SQL

- For XML Klausel

```
select * from table
for XML auto, elements
```

- Sybase-Beispiel

```
select XMLGEN( '<e>{$var}</e>', attr as var )
from table
```

- Oracle-Beispiel

```
for $book in sqlquery("select * from Book")/ROW
return $book/title
```

Die `for XML` Klausel ermöglicht eine xml- formatierte Ausgabe von SQL-Abfragen. Der Aufruf der SQL-Abfrage aus XQuery heraus kann über die `doc( uri )` Funktion erfolgen. Der uri enthält die SQL-Abfrage oder verweist auf ein Programm, das eine vordefinierte Abfrage ganz oder teilweise enthält.

Die Sybase-Variante benützt SQL als Hauptsprache. XQuery dient zur Konstruktion von XML-Fragmenten oder -Dokumenten innerhalb eines auszuliefernden Datensatzes. Die Abfrageresultate werden in einer Applikation als *Datensätze*, beispielsweise über *JDBC*, abgeholt.

Die Oracle-Variante benützt XQuery als Hauptsprache. Mit SQL-Abfragen werden Datensätze als Sequenz von XML-Elementen als Input für XQuery geliefert. Die Abfrageresultate werden in einem Java-Client über eine eigene Schnittstelle (OJXQI) als DOM-Knoten (Implementationsklasse `XMLNode` in Oracle) ausgeliefert. Die Funktion `sqlquery()` ist proprietär, entspricht aber der XQuery-Standardfunktion `collection()`. Diese Variante ist vielversprechend, da sie auf bestehenden Daten aufsetzt, die mächtigen Optimierungsmöglichkeiten von SQL zur Verfügung stellt, und gleichzeitig die Erzeugung von beliebigen XML-Strukturen zulässt. Anstelle der Funktion `sqlquery` hätte auch die Funktion `collection()` gemäss XQuery-Standard verwendet werden können.

Die drei Ansätze können natürlich kombiniert werden.

XQuery und Java, XQJ

```
import javax.xml.xquery.*;
...
XQDataSource ds = new IPSIXQDataSource();
XQConnection con = ds.getConnection();
try {
    String xquery = "...";
    XQExpression expr = con.createExpression();
    XQResultSequence seq = expr.executeQuery( xquery );
    while ( seq.next() ) {
        XQItem item = seq.getItem();
        Node n = item.getNode();
        ...
    }
} catch ( XQException e ) { ... }
```

→ Beilage: XQJ Demo

Dieses Beispiel arbeitet mit der XQJ-Spezifikation von Sun, die momentan (Dez. 04) in einem Working Draft Status ist. Die verwendete Implementation stammt aus dem IPSI XQuery Demonstrator der deutschen Fraunhofer Gesellschaft.

Das Abfrageresultat wird hier in Form einzelner DOM-Nodes abgeholt und kann so weiterverarbeitet werden.

III

XQuery im Detail

In diesem Kapitel wird auf den Umgang mit untypisierten XML-Daten eingegangen, das heisst solchen, die nicht durch ein XML Schema beschrieben sind.

Datentypen in XQuery

- XQuery ist eine typisierte Sprache. Grundsätzlich ist zu unterscheiden zwischen:
 - Atomaren Typen
 - Knotentypen
 - XQuery stellt in vielen Fällen automatische Typumwandlung zur Verfügung.
- Beilage: Typhierarchie von XQuery und XPath

Die von XML Schema ausgeliehenen Datentypen tragen das Präfix `xs:`

Die von XPath und XQuery definierten Datentypen tragen das Präfix `xdt:`

Mit Hilfe von XML Schema können auch *benutzerdefinierte* Typen erstellt werden.
Diese Möglichkeit soll im Moment hier nicht betrachtet werden.

Atomare Typen

```
let $x := 1  
→ $x enthält den atomaren Typ xs:integer
```

```
let $x := 1.11  
→ $x enthält den atomaren Typ xs:decimal
```

```
let $x := 1E7  
→ $x enthält den atomaren Typ xs:double
```

```
let $x := 'abc'  
→ $x enthält den atomaren Typ xs:string
```

```
let $x := true()  
→ $x enthält den atomaren Typ xs:boolean
```

Stringkonstanten dürfen auch in doppelten Anführungszeichen stehen und dürfen die eingebauten Entity-Referenzen `>`, `<`, `&`, `'` und `"` enthalten, sowie Zeichenreferenzen der Art `&#nnnnnn` enthalten.

Strings sind immer eine Folge von Unicode-Zeichen. Das Speicher-, Transfer- oder Ausgabe-Format für Zeichen ist meist UTF-8.

Der Typ `xs:decimal` ist der allgemeine, präzise Typ für Zahlen mit Kommastellen und entspricht damit dem SQL-Datentyp `numeric` respektive `decimal`. Ungewöhnlicherweise ist `xs:integer` ein von `xs:decimal` *abgeleiteter* Typ.

Konstruktion von atomaren Typen

```
let $x := xs:boolean( 'true' )
```

```
let $x := xs:dateTime( '2004-10-31T08:59:59.999' )
```

```
let $x := xs:integer( 123456789 )
```

```
let $x := xs:integer( '123456789' )
```

→ Typkonvertierung von String nach Integer

```
let $x := xs:duration( 'P1Y2M3DT4H5M6S' )
```

→ 1 Jahr, 2 Monate 3 Tage, 4 Stunden, 5 Min, 6 Sek

Für Zahlen können die speziellen Werte "NaN", "INF" und "-INF" angegeben werden.

Der Typ `xdm:untypedAtomic`

- Wenn aus einem XML-Knoten ein atomarer Wert gewonnen werden muss, so hat dieser den Typ `xdm:untypedAtomic`, es sei denn, der Typ ist durch eine zum Dokument gehörige XML Schema Beschreibung definiert.

`<a>123</a>` → 123 ist vom Typ `xdm:untypedAtomic`

- Dies ist bei Vergleichen relevant, weil `xdm:untypedAtomic` bei einem Vergleich mit einem anderen Typen in diesen anderen Typ umgewandelt wird.

`<a>123</a> = 123.0` → `true()`, weil Zahlenvergleich

`<a>123</a> = '123.0'` → `false()`, weil Stringvergleich

Wird aus einem XML-Element der Textinhalt extrahiert, so ist dies der ungetypte, rein textuelle Wert des Elementes (Nicht der Stringwert). Dieser textuelle Wert ist mit dem künstlichen Typ `xs:untypedAtomic` bezeichnet. Der ungetypte Wert wird in vielen Fällen wie ein String behandelt, kann jedoch beispielsweise auch in ein Datum umgewandelt werden, zwecks Vergleich mit einem Datumswert. Nur ungetypte Werte werden in bestimmten Situationen in einen anderen Typ umgewandelt.

Typabfragen

```
let $a := <x>Knoten X</x>
return $a instance of element()
```

```
let $a := ( <!-- Kommentar --> )
return ( $a instance of comment() )
```

```
declare default element namespace
    'http://sws.bfh.ch/catalog';
for $p in doc( 'catalog-mit-Schema.xml' )//pages
return fn:data($p) instance of xs:positiveInteger
```

→ Beilage: Beispieldokument catalog-mit-Schema-xml

→ Beilage: Beispieldokument Typ-Abfragen

Alle obigen Abfragen ergeben `true` als Resultat.

Eine *direkte* Typabfrage ist nicht möglich.

Typkonvertierung mit `cast as` und `treat as` sind möglich. `treat as` ist nur eine Zusicherung an den Compiler zur Kompilationszeit, dass Variablen keine anderen Belegungen als den mit `treat as` vorgesehenen Typ erhalten werden.

XML-Elemente sind typisiert, wenn sie mit einem XML Schema definiert wurden. XQuery behandelt dann ein XML-Element entsprechend seinem Typ im XML Schema.

Operationen auf atomaren Typen

- Arithmetische Operatoren

`+ - * div idiv mod`

- Logische Operatoren

`and or not`

- Operationen für Strings sind über Funktionen verfügbar

`concat( 'a', 'b', 'cde', 'efg' )`

`substring( 'abc', 1, 3 )`

`string-length( 'abc' )`

→ Beilage: Funktionen und Operatoren

Der Schrägstrich / hat eine spezielle Bedeutung in XQuery, deshalb werden die Divisionsoperationen mit den Worten `div` und `idiv` ausgedrückt.

Zu Strings und vorallem auch zu Datums- und Zeittypen existieren eine Vielzahl von Funktionen.

Vergleich von atomaren Typen

- Atomare Typen können mit den üblichen Operatoren verglichen werden. Wenn einer der Operatoren vom Typ `xdt:untypedAtomic` ist, wird er in einen dem anderen Operanden entsprechenden Typ umgewandelt. Ansonsten müssen beide Typen vergleichbar sein.
- Operatoren für Wertevergleich: `eq`, `ne`, `lt`, `gt`, `ge`, `le`

`'abc' eq 'abc' → true()`

`1.00 eq 1 → true()`

`1.0 eq '1.0' → Laufzeitfehler wegen Typvergleich`

Knotenarten

- Es gibt 7 Knotenarten in XQuery, entsprechend der Idee von XML:
 - Dokumentknoten oberster Knoten eines Dok.
 - Elementknoten `<e>text</e>`
 - Attributknoten `<e a='a'></e>`
 - Textknoten `<e>text</e>`
 - Kommentarknoten `<!-- kommentar -->`
 - Verarbeitungsanweisungen `<?target name ?>`
 - Namensraumknoten `<e xmlns:A >text</e>`
- Knoten sind nicht Werte, sondern haben die Semantik von Objekten

Konstruktion von Knoten

- Elementknoten können ohne Anführungszeichen geschrieben werden:

```
let $x := <a>ich bin ein XML Element</a>
```

- Ein Element und sein Name können dynamisch erzeugt werden:

```
let $e := 'a'  
let $x := element {$e} {'ich bin ein XML Element'}  
return $x
```

```
let $e := 'elem'  
let $a := 'attr'  
let $x := element {$e} {  
    ( attribute{$a}{'ainhalt'}, 'einhalten' ) }  
return $x
```

Ausgabe von `return`?

Atomarisierung

- Für gewisse Aufgaben wird oft nur der Inhalt eines Knoten, dessen Wert benötigt.
- Die Extraktion eines Knotenwertes wird aus Komfortgründen an vielen Stellen durch XQuery automatisch vorgenommen und heisst Atomarisierung.
- **Atomarisierung** findet beispielsweise in folgenden Ausdrücken statt:

<code><a>hello eq hello</code>	<code>→ true</code>
<code><a>hello eq 'hello'</code>	<code>→ true</code>
<code><a>1 + 1</code>	<code>→ 2</code>
<code><a>12 + 1</code>	<code>→ 13</code>

→ Beilage: Atomarisierung

Atomisierung und automatische Typkonversion gehen oft Hand in Hand wie obige Beispiele zeigen.

Vergleich von Knoten (1)

- Atomisierter Wertevergleich mit `eq` oder `=`

```
<a>hello</a> eq <b>hello</b>      ➔ true
```

```
<a>hello</a> = <b>hello</b>        ➔ true
```

- Identitätsvergleich mit `is`

```
let $x1 := <a>1</a>
```

```
let $x2 := <a>1</a>
```

```
return $x1 is $x2                  ➔ false
```

- Positionsvergleich mit `<<` und `>>`

Ein Knoten ist kleiner als ein anderer, wenn seine Position in der Dokumentordnung kleiner ist.

Die Dokumentordnung entspricht einer Numerierung der Knoten in einem Dokument, wenn man vom Wurzelknoten ausgehend rekursiv alle Kindknoten besucht, zuerst in die Tiefe, dann in die Breite fortschreitend.

Vergleich von Knoten (2)

- Ein präziser Vergleich von Knoten, der sowohl die Struktur wie den Inhalt berücksichtigt, wird mit `deep-equal()` realisiert.
- Auf jeder Ebene eines komplexen Knotens müssen Art des Knotens, der Name und der Inhalt identisch sein.
- Enthält eine Knoten mehrere Kindknoten, müssen diese in Art, Reihenfolge, Name und Wert übereinstimmen.

→ Beilage: Gleichheit von Knoten

Sequenzen

- Sequenzen sind die zentrale Datenstruktur in XQuery.
- eine Sequenz hat 0 bis n Items.
- ein Item ist ein atomarer Wert oder ein Knoten.
- eine Sequenz kann Items *unterschiedlichen* Typs enthalten.
- Eine Sequenz mit *einem* Eintrag ist identisch mit dem Eintrag selbst.
- Sequenzen sind *geordnet*.
- Sequenzen werden als komma-separierte Liste dargestellt, eingefasst in Klammern.
- Es gibt keine verschachtelten Sequenzen.
- Ein XPath-Ausdruck liefert eine Sequenz zurück.

Beispiele von Sequenzen

gültige Sequenzen:

```
()  
( 1 )  
( 1, 2, 3 )  
( 'a', 'b', 'c' )  
( <a/>, <b/>, <c><d></d></c> )  
( 'hello', 3.8, xs:date('2004-01-31') )  
( 'a', 'b', 'c', ( 'a', 'b', 'c' ) )
```

ungültige Sequenz:

```
( <a/><a/>, 2, 3 )
```

Ein-elementige Sequenzen

- Eine Sequenz aus *einem* Eintrag ist identisch mit dem Eintrag selbst. Folgende Abfragen mit dem Vergleichsoperator = oder eq liefert true zurück für alle Belegungen von \$x.

(\$x) = \$x → true

(\$x) eq \$x → true

- Ein einzelner Wert oder Knoten ist identisch mit einer ein-elementigen Sequenz aus diesem Wert oder Knoten.

Obige Vergleiche gelten sowohl für Knoten wie für atomare Werte von \$x. Bei Knoten ist noch zu berücksichtigen, dass für den Vergleich eine Atomisierung stattfindet. Das heisst, die Knoten werden durch ihre Werte ersetzt, also beispielsweise <a>123 durch 123. Das bedeutet im Weiteren, dass beispielsweise der Vergleich <a>123 = 123 den Wert true ergibt.

Verschachtelte Sequenzen

- Sequenzen können *nicht* verschachtelt werden.
- Eine verschachtelte Eingabe ist möglich, wird aber wie eine sequentielle Eingabe behandelt. Es gilt also beispielsweise:

(x , (x , x)) ist identisch mit (x , x , x)
(x , ()) ist identisch mit (x)

Zugriff auf einzelne Sequenzeinträge

- Eintrag an erster Position entnehmen

```
let $x := ( 'a', 'b', 'c' )  
return $x[1]
```

- Eintrag an letzter Position entnehmen

```
let $x := ( 'a', 'b', 'c' )  
return $x[last()]
```

- Neue Sequenz aus den zwei letzten Einträgen bilden

```
let $x := ( 'a', 'b', 'c' )  
return $x[position() > 1 and position() <= 3]
```

Gleichheit von Sequenzen

- Zwei Sequenzen `$S1` und `$S2` sind gleich, wenn
 - beide Sequenzen die gleiche Anzahl Einträge enthalten
 - für jede Position `i` der Vergleich
`deep-equal( $S1[i], $S2[i] )`
den Wert `true` ergibt
- Zwei leere Sequenzen sind immer gleich
- Der Vergleich von Sequenzen wird mit `deep-equal()` ausgeführt, also beispielsweise `deep-equal( $S1, $S2 )`

Handelt es sich bei einzelnen Einträgen der beiden Sequenzen um atomare Werte, ist der Vergleich mit `deep-equal()` und mit dem Operator `eq` gleichwertig.

Allgemeine Vergleiche für Sequenzen

- Für den Vergleich von Sequenzen gibt es auch die Operatoren =, !=, >, <, >=, <=
- Die Operatoren sind nützlich, aber auch überraschend:

```
( 1,2,3 ) = ( 3,4,5 )    ➔ true !  
( 1,2,3 ) != ( 1,2,3 ) ➔ true !  
( 1,2,3 ) < ( 1,2,3 )  ➔ true !  
( 1 )      = ( <a>1</a> ) ➔ true !
```

Erklärung: Es gibt mindestens *ein* Paar von Werten, die den Vergleich erfüllen.

- Für Knoten in der Sequenz findet eine Atomisierung statt.

Da eine Sequenz mit einem Eintrag gleich dem Eintrag selbst ist, gelten diese Vergleiche auch für einzelne Werte.

Test auf leere Sequenz

Mit der Funktion `exists()` kann eine Sequenz auf leer getestet werden. Dies ist insbesondere nützlich, wenn bestimmte Elemente in einem XML-Dokument optional sind.

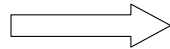
```
for $b in doc( "catalog-generic.xml" )//book  
return fn:exists( $b/subtitle )
```

Siehe auch Beispiel zu `if then else`

Ausgabe von Sequenzen, Serialisierung

- In vielen Fällen wird das Resultat einer XQuery-Abfrage als Zeichenfolge auszugeben sein. Dieser Prozess heisst Serialisierung.

```
let $x := ( 1, 2, <a/> )  
return $x
```



```
1 2 <a/>
```

- Sequenzeinträge werden durch ein Leerzeichen abgetrennt.
- Attribut- und Namespace-Knoten können nicht serialisiert werden.

→ Beilage: Serialisierung

FLWOR Ausdrücke

- Ein FLWOR-Ausdruck ist das augenfälligste Konstrukt in XQuery und steht für folgende Angaben:

FOR	→ eine Schleife über eine Sequenz von Knoten/Werten
LET	→ eine Zuweisung pro Schlaufendurchgang
WHERE	→ eine Bedingung, ausgewertet pro Schlaufendurchgang
ORDER BY	→ eine Sortierung über alle aus der Schleife resultierenden Knoten/Werte
RETURN	→ eine Ausgabeanweisung pro Knoten/Wert in der aus FLWO resultierenden Sequenz

→ Beilage: FLWOR Ausdrücke

Wie aus dem Grammatik-Ausschnitt in der Beilage ersichtlich ist, bestehen sehr umfangreich Verschachtelungsmöglichkeiten für FLWOR-Ausdrücke.

FLWOR-Beispiele (1)

```
for $book in doc( "catalog-generic.xml" )//book  
return $book/title
```

```
let $catalog := doc( "catalog-generic.xml" )  
for $book in $catalog//book  
where $book/title = 'Learning XML'  
return $book/isbn
```

```
for $book in doc( "catalog-generic.xml" )//book at $i  
return ( $i, $book )
```

```
for $i in ( 1 to 100 )  
return $i
```

→ Beilage: Beispieldokument catalog-generic.xml

FLWOR-Beispiele (2)

```
for $book in doc( "catalog-generic.xml" )//book
order by $book/subtitle descending empty least
return $book
```

```
for $book in doc( "catalog-generic.xml" )//book
let $numAuthors := fn:count( $book//author )
return ( $book/title/text(), $numAuthors )
```

Sortieren: jedes Kriterium kann mit `ascending` oder `descending` versehen werden. Ausserdem können leere Werte durch `empty least` oder `empty greatest` als niederstwertig oder als grösstwertig behandelt werden.

Mehrere FOR-Variablen, Joins

- Verwendet man mehrere FOR-Variablen entstehen (konzeptionell gesehen) Mehrfachschlaufen.
- Die Verwendung ist analog wie bei Joins in SQL.

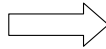
```
for $book in doc( "catalog-generic.xml" )//book,  
    $res in doc( "reservations.xml" )//reservation  
where $book/isbn = $res/book/isbn  
return( $res/person/email, $book/title )
```

- Beilage: Beispieldokument catalog-generic.xml
- Beilage: Beispieldokument reservations.xml

Mischung XML und XQuery

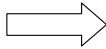
- Literales XML und XQuery-Ausdrücke dürfen gemischt verwendet werden. Es ist darauf zu achten, dass der XQuery-Parser ausgeschaltet wird, wenn er auf gültiges XML trifft. Er ist also jedesmal mit `{ }` wieder einzuschalten.

```
for $z in ( 1 to 3 )  
return <zahl>$z</zahl>
```



```
<zahl>$z</zahl>  
<zahl>$z</zahl>  
<zahl>$z</zahl>
```

```
for $z in ( 1 to 3 )  
return <zahl>{$z}</zahl>
```



```
<zahl>1</zahl>  
<zahl>2</zahl>  
<zahl>3</zahl>
```

XHTML-Ausgabe

- Ein XQuery Abfrageresultat kann ohne Weiteres XHTML-Format haben:

```
<html><body>
<h1>Buchangebote über XML</h1>
{
  for $b in $input/book
  where contains( $b/title, 'XML' )
  order by $b/title
  return <h2>{$b/title/text()}</h2>
}
</body></html>
```

- Anwendungsmöglichkeiten: Reporting-Tools

Pattern Matching, reguläre Ausdrücke

- Patternmatching mit regulären Ausdrücken ist unabdingbar für das Suchen in und nach Strings.
- XQuery lehnt sich an die bekannte Perl-Syntax an.
- Beispiel: Suche alle Bücher mit dem Begriff "nutshell" im Titel, unabhängig von der Gross-/Kleinschreibung.

```
for $book in doc( "catalog-generic.xml" )//book
where fn:matches( $book/title, '.*nutshell.*', 'i' )
return $book
```

Für die verbindliche Syntax für reguläre Ausdrücke in XQuery wird auf 'XML Schema Part 2: Datatypes Second Edition' des W3C verwiesen.

Es gelten folgende einfachen Musterbeschreibungen

<i>(zeichenfolge)*</i>	kein bis beliebiges Auftreten von <i>zeichenfolge</i>
<i>(zeichenfolge)+</i>	mindestens einmaliges Auftreten von <i>zeichenfolge</i>
<i>(zeichenfolge)?</i>	höchstens einmaliges Auftreten von <i>zeichenfolge</i>
<i>(zeichenfolge){n,m}</i>	n- bis m-maliges Auftreten von <i>zeichenfolge</i>
<i>(A) (B)</i>	Ausdruck A oder B
<i>()</i>	Klammerung
<i>[abc]</i>	das Zeichen a, b oder c
<i>[^abc]</i>	jedes Zeichen ausser a, b oder c
<i>.</i>	beliebiges Zeichen
<i>\s</i>	beliebiges Leerraumzeichen

Im Gegensatz zu Perl-Ausdrücken sind XQuery-reguläre-Ausdrücke immer mit dem Anfang respektive dem Ende des zu testenden Strings verankert:

- das Muster *abc* in XQuery ist identisch zum Muster *^abc\$* in Perl
- das Muster *.*abc.** in XQuery ist identisch zum Muster *abc* in Perl

Zwei weitere Funktionen, *fn:replace()* und *fn:tokenize()* ermöglichen das Ersetzen von regulären Ausdrücken in Strings, respektive das Aufspalten von Strings mit einem regulären Ausdruck als Trennmuster.

Der dritte Parameter enthält allgemeine Flags für das Patternmatching. Ein *'i'* zeigt an, dass das Matching unabhängig von der Gross-/Kleinschreibung ist.

Aggregatfunktionen

- Analog zu SQL stehen die in folgendem Beispiel gezeigten Aggregatoperationen zur Verfügung. Soweit anwendbar, stehen diese Operationen für Strings, Zahlen und weitere Datentypen zur Verfügung.

```
let $books := doc( "catalog-generic.xml" )//book
return ( avg( $books/pages ),
        min( $books/pages ),
        max( $books/pages ),
        sum( $books/pages ),
        count( $books ) )
```

Gruppierung (1)

- Eine explizite Gruppierung mit `group by` existiert nicht und ist auch nicht notwendig. Die vorgängig erwähnten Aggregatfunktionen eignen sich aber gut für die Gruppierung.
- Beispiel 1 mit Gruppierung nach *Struktur*.
Suche die Anzahl Subelemente von jedem `<book>` Element:

```
let $books := doc( "catalog-generic.xml" )//book
for $b in $books
return count( $b/* )
```

Das zentrale Konstrukt in dieser Abfrage ist das Ausführen eines XQuery-Ausdrucks innerhalb eines Funktionsaufrufes.

Gruppierung (2)

- Beispiel 2 mit Gruppierung nach *Wert*.
Suche die Anzahl Bücher gruppiert nach Verlag:

```
let $books := doc( "catalog-generic.xml" )//book
let $pubs  := doc( "catalog-generic.xml" )//publisher
for $p in distinct-values( $pubs )
return ( $p,
count( for $b in $books where $b/publisher = $p return $b )
)
```

Für die Gruppierung nach Werten ist die Funktion `distinct-values()` von entscheidender Bedeutung.

Die Ausgabe des Verlagsnamens ist der Einfachheit halber in diese Abfrage nicht integriert.

intersect, except, union

- Diese Mengenoperationen können praktisch sein und eine intuitive Abfragedarstellung ergeben. Grundsätzlich sind sie auch ersetzbar durch `and-`, `and not-` oder `or`-Konstrukte.
- Beispiel: Suche alle Bücher von Eliotte Rusty Harold, aber nicht von W. Scott Means.

```
( let $books := doc( "catalog-generic.xml" )//book
for $b in $books where $b/author = 'Eliotte Rusty Harold'
return $b )
except
( let $books := doc( "catalog-generic.xml" )//book
for $b in $books where $b/author = 'W. Scott Means'
return $b )
```

Zu beachten ist, dass für diese drei Operationen die beteiligten Sequenzen als Mengen betrachtet, das heisst Duplikat-Einträge entfernt werden.

Teilbäume abschneiden

- Beispiel: Alle Bücher die neuer als 2000 sind. Aus den zurückgelieferten Angaben jedoch das Element pages herausschneiden.

```
for $book in doc( "catalog-generic.xml" )//book
return
  <book>
    {
      $book/* except $book/pages
    }
  </book>
```

- Das Herausschneiden von Teilbäumen ist nicht sehr stark entwickelt.

Das Herausschneiden von Teilbäumen ist sehr oft mit dem Begriff *Subtree Pruning* beschrieben. Einfache Konstrukte wie oben dargestellt sind möglich. Komplexe Abfragen wie 'Gib alle `book`-Elemente zurück, schneide aber in beliebiger Tiefe die Elemente mit dem Namen `keyword` heraus' sind nicht möglich.

Quantoren (1)

- Alle Bücher, die unterem anderem von E. R. Harold geschrieben sind:

```
for $book in doc( "catalog-generic.xml" )//book
where some $author in $book/author
      satisfies $author = 'Elliotte Rusty Harold'
return $book
```

Die erste Abfrage ist identisch zu:

```
for $book in doc( "examples/catalog-generic.xml" )//book
where $book/author = 'Elliotte Rusty Harold'
return $book
```

Quantoren (2)

- Alle Bücher, die ausschliesslich von E. R. Harold geschrieben sind:

```
for $book in doc( "catalog-generic.xml" )//book
where every $author in $book/author
      satisfies $author = 'Elliotte Rusty Harold'
return $book
```

Die erste Abfrage ist identisch zu:

```
for $book in doc( "examples/catalog-generic.xml" )//book
where $book/author = 'Elliotte Rusty Harold'
return $book
```

if then else

- Ausgabe aller Untertitel. Wenn kein Untertitel vorhanden ist, dann Ausgabe von 'Kein Untertitel'

```
for $book in doc( "catalog-generic.xml" )//book
return
  if ( fn:exists( $book/subtitle ) )
  then $book/subtitle
  else <subtitle>Kein Untertitel</subtitle>
```

Benutzerdefinierte Funktionen

- Beispiel

```
declare function local:zaehler( $a as item()* )
as xs:integer
{
    count( $a )
};
```

```
let $x := doc( "catalog-generic.xml" )//book
return local:zaehler( $x )
```

- Funktionen dürfen rekursiv aufgerufen werden.

Für Funktionen muss ein Namensraum angegeben werden. Es ist von XQuery ein Namensraum local: vordefiniert, damit er nicht noch eingeführt werden muss.

Module

Moduldatei myMod.xquery :

```
module namespace myMod = "http://www.bfh.ch/myMod";  
declare function myMod:myFunc1() { 'Funktion 1' };  
declare function myMod:myFunc2() { 'Funktion 2' };
```

XQuery-Abfrage

```
import module namespace m = http://www.bfh.ch/myMod'  
at 'examples/myMod.xquery';  
let $s := m:myFunc1()  
return $s
```

Zu beachten: die aktuelle Namespace-Abkürzung im import-Befehl muss nicht mit der Namespace-Abkürzung in der Modul-Datei übereinstimmen, der volle Namespace-URL hingegen schon.

XQuery Prolog

- Der Prolog ist eine Serie von Angaben zu:
 - Variablendefinitionen
 - Namensraumdefinitionen
 - Schemaimport
 - Modulimport
 - Sortierordnung
 - Leerraumbehandlung
 - Benutzerdefinierte Funktionen

Die Sortierordnung ist normalerweise eingestellt auf <http://www.w3.org/2004/10/xpath-functions/collation/codepoint>. Diese erzeugt beispielsweise folgende Reihenfolge für die genannten Buchstaben:
a A à À ä Ä ... z Z

Eine Demo für die Unicode Standardsortierung ist zu finden unter <http://www.unicode.org/reports/tr10/Sample/>

Variablendeklaration

- Globale Variablen, welche in allen nachfolgenden XQuery-Ausdrücken bekannt sind.
- Globale Variablen eignen für die Aufnahme von Konstanten oder für Übergabe von Werten aus Programmiersprachen heraus.

```
declare variable $pi as xs:decimal := 3.14159;  
for $i in ( 1 to 10 )  
return $pi * $i
```

```
declare variable $isbn as xs:string external;  
for $b in doc('catalog-generic.xml')//book  
where $b/isbn = $isbn  
return $b
```

XQuery Full-Text

- XML eignet sich sehr gut zur Ablage von halbstrukturiertem oder freiem Text.
- XQuery Full-Text bietet die Möglichkeit, in längeren Textabschnitten nach Worten, Wortfolgen, Wortmustern zu suchen.
- XQuery Full-Text gliedert hierzu ein XML-Dokument logisch in Sätze, Abschnitte und Worte.
- Bei der Abfrage können Abstand, Reihenfolge, Stopwörter, WortStamm, boolesche Verknüpfungen usw. berücksichtigt werden.
- Die Suchresultate können mit einer score()-Funktion gewichtet werden.

→ Beilage: XQuery Full-Text

XQuery Full-Text Implementationen sind noch rar. Beispiel:
http://www.galaxquery.com/galatex/demo/galax_demo.html

IV

Zusammenfassung und Ausblick

Zusammenfassung

- XQuery ist eine vielversprechende Abfragsprache für XML-strukturierte Daten, analog zu SQL für relationale Datenbanken.
- Die Produktentwicklung ist in vollem Gang.
- Die Verabschiedung als W3C Recommendation steht noch aus.
- Ein Sprachteil für das Ändern von Daten ist noch nicht in Sicht.

Übersicht Zugriffstechnologien

