7 Suchen auf XML-Daten

7.1 Semistrukturierte Daten in XML

7.2 Boolesches Retrieval mit XML-QL

7.3 Erweiterung von XML-QL um Keyword-Suche

7.4 XXL: Ranked Retrieval auf XML-Daten

Literatur: S. Abiteboul, P. Buneman, D. Suciu,
Data on the Web: From Relations to Semistructured Data and XML, Morgan Kaufmann, 1999

7.1 Semistrukturierte Daten in XML

XML (Extensible Markup Language) ist **der** (W3C) Standard zur Datenrepräsentation für Datenaustausch, Datenintegration, Datentransformation, E-Commerce (B2B), usw.

Im Gegensatz zu HTML beschreibt XML nur die logische Struktur von Daten; Layout und Präsentation werden durch Style Sheets separat beschrieben (in XSL).

Historische Entwicklung:

SGML
↓
HTML
↓
XML

XML-zentrierte Technologien:

Definition/Struktur: DTD, XML-Schema
Hyperlinks: Xlink, XPointer
Layout/Filtern: XSL, CSS
Anfragesprachen:
XQL, XPATH, XML-QL, XSLT, Quilt, ...
Metadaten: RDF, ...
APIs: DOM, SAX
Spezialisierungen
(z.B. domainspezifische Ontologien):
MathML, Rosetta, ...

Grundkonzepte von XML

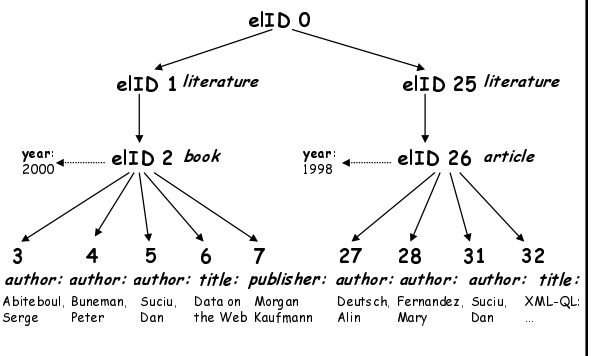
- (Frei definierbare) Tags: book, title, author
  - mit Start-Tag: <book> usw.
  - und End-Tag: </book> usw.
- **Elemente**: <book> ... </book>
  Elemente haben einen Namen (book) und einen Inhalt (...)
  Elemente können geschachtelt werden
- Jedes XML-Dokument hat ein Wurzelelement und bildet einen Baum

Elementinhalte können, müssen aber nicht typisiert sein (meist (P)CDATA, also Strings).
Elemente können **Attribute** haben, die jeweils einen Namen und einen Wert (Inhalt) haben, z.B. <article year=1999>.
Elemente haben optional Id-Attribute (Element-Ids), auf die innerhalb eines Dokuments mit dem Attribut idref verwiesen werden kann.
Elemente können Hyperlinks als Attribute href haben.

XML-Beispiel

```
<literature>
<book year="2000">
    <author>Abiteboul, Serge</author>
    <author>Buneman, Peter</author>
    <author> Suciu, Dan</author>
    <title>Data on the Web</title>
    <publisher>Morgan Kaufmann</publisher>
  </book>
  ...
  <article year="1998">
    <author>Deutsch, Alin</author>
    <author>Fernandez, Mary</author>
    ...
    <author>Suciu, Dan</author>
    <title>XML-QL: A Query Language for XML</title>
  </article>
  ...
</literature>
```


XML-Daten als markierte Graphen (mit Elementnamen als Knoten-Labels)



Beispiel für Referenzen in XML

```
<person id="o555">
   <name>Jane</name>
   <children idref="o111 o222"/>
</person>
<person id="o666">
   <name>Tarzan</name>
   <children idref="o111 o222"/>
   <homepage href="http://www.tarzan.biz/home.htm"></homepage>
   <bizportal href="http://www.tarzan.biz/portal.xml"/>
</person>
<person id="o111">
   <name>BillyBoy</name>
   <mother idref="o555"/> <father idref="o666"/>
</person>
<person id="o111">
   <name>Barbie</name>
   <mother idref="o555"/> <father idref="o666"/>
</person>
```

DTDs zur XML-Strukturierung

Ein XML-Dokument heißt **wohlgeformt**, wenn seine Tagstruktur einer korrekten Klammerung entspricht.
Eine **DTD (Document Type Definition)** ist eine kontextfreie Grammatik zur Beschreibung erlaubter Tagstrukturen (und damit zur Strukturierung von XML-Dokumenten eines Typs).
Ein XML-Dokument heißt **gültig** bzgl. einer DTD wenn seine Tagstruktur durch die DTD generierbar ist.

Beispiel einer DTD:

```
<!DOCTYPE book [
  <!ELEMENT book (title, author*, publisher?, section+)>
  <!ATTLIST book year CDATA #IMPLIED>
  <!ELEMENT title (#PCDATA)>
  <!ELEMENT author (#PCDATA)>
  <!ELEMENT section (#PCDATA | title | section)*>          ]>
```

Weitergehende Schematisierung/Typisierung möglich mit XML-Schema

7.2 Boolesches Retrieval mit XML-QL

XML-QL ist eine (von mehreren) Anfragesprache(n) für XML-Daten, die zur Diskussion für die W3C-Standardisierung steht.
Kombination von
- **logischen Bedingungen** über Element- und Attributinhalte (Prädikatenlogik 1. Ordnung a la SQL)
- **regulären Ausdrücken** zum Pattern-Matching von Elementnamen entlang von Pfaden im XML-Datengraph

+ Gruppierung, Aggregation, Strukturtransformation, usw.

Im Gegensatz zu Datenbanksprachen wie SQL bezieht man sich nicht notwendigerweise auf ein festes Strukturschema bzw. muß dieses nicht kennen.
Ein **Anfrageergebnis** ist eine Menge von sich qualifizierenden Pfaden oder Teilgraphen des XML-Datengraphen bzw. daraus konstruierten XML-Dokumenten.

Exkurs: Anfragesprache SQL auf Relationen

Bücher

ISBN	Autor	Titel	...	Kategorie
1111	Grass	Blechtrommel		Roman
2222	Gates	Robin Hood		Märchen
3333	King	Windows 2000		Horror

DB: Relationen (Tables)

Ausprägung

$\subseteq$ *Dom (ISBN)* $\times$ *Dom (Autor)* $\times$ ... *Dom (Kategorie)*

Verkäufe

ISBN	KNr	Datum	...
3333	999	1.4.00	...

Kunden

KNr	Name	Ort	PLZ	Strasse	...
901	Becker	Saarlouis	...		
902	Caesar	Rom			

Schema

Attribut (Column)

Tupel (Record, Row)

Konzepte von SQL (Structured Query Language)

Anweisungen zum **Suchen in Tabellen,**
Ändern von Tabellen,
Anlegen neuer Tabellen,
Integritätssicherung, Zugriffskontrolle, etc.

„stand-alone" und als API (z.B. in ODBC oder JDBC)

Grobsyntax von SQL-Abfragen (Queries):

```
SELECT (column | expression) { , (column | expression)}
FROM table [correlation_var] {, table [correlation_var]}
[ WHERE search_condition ]
[ GROUP BY column {, column}
   [ HAVING search_condition] ]
```


Beispiele für SQL-Abfragen

1) **Kunden aus Saarbrücken, die Bücher der Kategorie Politik gekauft haben**

```
SELECT K.KNr, K.Name
FROM Kunden K, Verkäufe V, Bücher B
WHERE K.Ort = „Saarbrücken"
AND K.KNr = V.KNr AND V.ISBN = B.ISBN
AND B.Kategorie = „Politik"
```

2) **Kunden aus Saarbrücken, die noch nie ein Buch der Kategorie Humor gekauft haben**

```
SELECT K.KNr, K.Name
FROM Kunden K
WHERE K.Ort = „Saarbrücken"
AND NOT EXISTS (
    SELECT B.ISBN FROM Bücher B, Verkäufe V
    WHERE B.Kategorie = „Humor"
    AND B.ISBN = V.ISBN AND V.KNr = K.KNr )
```


Semantik von SQL-Abfragen

μ: SQL $\rightarrow$ TRK TRK = Tupelrelationenkalkül $\approx$ Prädikatenlogik 1. Ordnung

mit

- μ [SELECT A, B, ... FROM R, S, ..., T, U, ... WHERE F]
  (wobei A, B, ... Attribute von R, S, ..., nicht aber von T, U, ...)
  $= \{r.A, s.B, ... \mid r \in R \wedge s \in S \wedge ...$
  $\wedge \exists t \exists u ... (t \in T \wedge u \in U \wedge ... \wedge \mu'[F])$
- μ' [F AND G] $= \mu'[F] \wedge \mu'[G]$
- μ' [F OR G] $= \mu'[F] \vee \mu'[G]$
- μ' [NOT F] $= \neg \mu'[F]$
- μ' [A θ B] = x.A θ y.B
- μ' [EXISTS S] (wobei S: SELECT D FROM P, Q, ... WHERE H)
  $= \exists p \exists q ... (p \in P \wedge q \in Q \wedge ... \wedge \mu'[H])$

Beispiel für SQL-Semantik

```
SELECT KNr, Name
FROM Kunden K
WHERE K.Ort = „Saarbrücken"
AND NOT EXISTS (SELECT B.ISBN FROM Bücher B, Verkäufe V
                WHERE B.Kategorie = „Humor"
                AND B.ISBN = V.ISBN AND V.KNr = K.KNr )
```

⇩ μ

$\{k.KNr, k.Name \mid k \in Kunden \wedge \mu'[K.Ort = ... \text{ AND NOT EXISTS } ...]\}$

$= \{k.KNr, k.Name \mid k \in Kunden \wedge \mu'[K.Ort = ...] \wedge \neg\ \mu'[EXISTS ...]\}$

$= \{k.KNr, k.Name \mid k \in Kunden \wedge k.Ort = ... \wedge \neg\ \mu'[EXISTS ...]\}$

$= \{k.KNr, k.Name \mid k \in Kunden \wedge k.Ort = ... \wedge \neg$
$(\exists b \exists v (b \in Bücher \wedge v \in Verkäufe \wedge$
$\mu'[B.Kategorie = ... \text{ AND } ... \text{ AND } ...])\}$

$= \{k.KNr, k.Name \mid k \in Kunden \wedge k.Ort = ... \wedge \neg$
$(\exists b \exists v (b \in Bücher \wedge v \in Verkäufe \wedge$
$b.Kategorie = ... \wedge b.ISBN = v.ISBN \wedge v.KNr = k.KNr))\}$

Gruppierung und Aggregation in SQL

Beispiel: Verkaufszahlen der Buchkategorien

```
SELECT B.Kategorie, SUM(V.Anzahl) FROM Bücher B, Verkäufe V
WHERE B.ISBN = V.ISBN GROUP BY B.Kategorie
```

ISBN	Autor	Titel	...	Kategorie
1111	Grass	Blechtrommel		Roman
2222	Gates	Robin Hood		Märchen
3333	Mann	Zauberberg		Roman

ISBN	...	Anzahl
3333	...	1
1111	...	1
2222	...	5
3333	...	2

ISBN	Kategorie	Anzahl
3333	Roman	1
1111	Roman	1
2222	Märchen	5
3333	Roman	2

ISBN	Kategorie	Anzahl
2222	Märchen	5
3333	Roman	1
1111	Roman	1
3333	Roman	2

Kategorie	Anzahl
Märchen	5
Roman	4

Konzepte von XML-QL

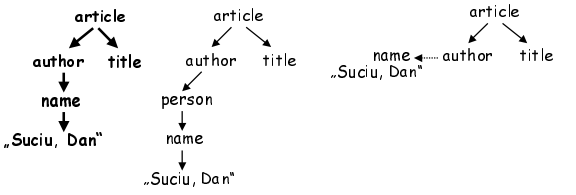
Eine Anfrage besteht aus drei Blöcken:
Where - In – Construct (≈ Where-From-Select)

Pfadausdrücke:
Elementnamen entlang eines Pfades im XML-Datengraphen
z.B.: <article.author.name>
mit Wildcards bzw. regulären Ausdrücken
z.B. <article.*.name> oder <article*.author>

Elementvariable:
werden an den Elementnamen und –inhalt
am Ende eines Pfades gebunden
z.B.: <author.name> $n </>
oder an das gesamte Element (mit Unterelementen)
z.B.: <author.name> </> ELEMENT_AS $n
und können dann mehrfach verwendet werden:
- zur Spezifikation von Filterbedingungen
  z.B. $n Like „%Suciu%" And ...
- zur Konstruktion der Resultatstruktur

XML-QL-Beispiel (1)

```
// suche nach Web-bezogenen Artikeln von Dan Suciu aus dem Jahr 1998
WHERE <article year=„1998">
        <author.name>$n</>
        <title>$t</>
      </>
      IN „www.books/bib.xml", $n LIKE „Suciu%", $t LIKE „%web%"
CONSTRUCT <result>$n $t</>
```

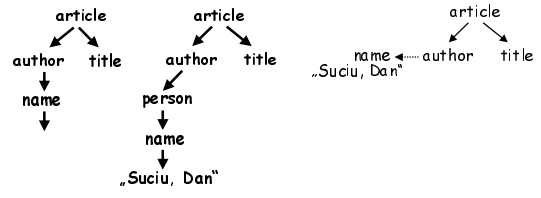


Treffer *kein Treffer* *kein Treffer*

XML-QL-Beispiel (2)

```
// suche nach Web-bezogenen Artikeln von Dan Suciu aus dem Jahr 1998,
wobei „name" ein Subelement in beliebiger Tiefe sein kann

WHERE <article year=„1998">
        <author.*.name>$n</>
        <title>$t</>
      </>
      IN „www.books/bib.xml", $n LIKE „Suciu%", $t LIKE „%web%"
CONSTRUCT <result>$n $t</>
```



Treffer Treffer *kein Treffer*

XML-QL: Elementvariable

```
WHERE <$p>
        <title>$t</>
        <$e>Suciu, Dan</>
      </>
      IN „www.books/literature.xml",
      $p IN {article, book}
CONSTRUCT <$p>
            <title>$t</>
            <$e>Suciu, Dan</>
          </>
```

XML-QL: Pfadausdrücke

Fragment eines XML-Dokuments:

```
<part>
  <part>
    <part>
      <name>Escort</>
      <brand>Ford</>
    </>
  </>
</>
```

(a)
```
WHERE <part*>
            <name>$n</>
            <brand>Ford</>
          </>
          IN „www.books/autos.xml"
CONSTRUCT <result>
                <name>$n</>
              </>
```

(b)
```
WHERE <part+.(name|brand)>$n</>
          IN „www.books/autos.xml"
CONSTRUCT <result>
                <name>$n</>
              </>
```


XML-QL: Anfragen über mehrere Dokumente (Joins)

```
// Artikel von Autoren, die ein Buch nach 1995 geschrieben haben

WHERE <article>
            <author>
              <firstname> $f </> <lastname> $l </>
            </></> CONTENT_AS $a
          IN „www.a.b.c/bib.xml",
          <book year=$y>
            <author>
              <firstname> $f </> <lastname> $l </>
          </></>
          IN „www.x.y.z/bib.xml",

          y > 1995

CONSTRUCT <article> $a </>
```


XML-QL: weitere Features

```
// suche nach „name" als Attribut, in beliebiger Tiefe

WHERE <article year=„1998">
            <author.*  name=$n></>
            <title>$t</>
          </>
          IN „www.books/bib.xml", $n LIKE „Suciu%", $t LIKE „%web%"
CONSTRUCT <result>$n $t</>
```

+ Gruppieren mit geschachtelten Queries
+ Join über Elementnamen
...

Speicherung XML-Daten in relationaler DB

Ein Ansatz (von mehreren):
Für jeden Element- oder Attributtyp eine ternäre bzw. binäre Relation, die die von Elementen des entsprechenden Typs ausgehenden Kanten enthält sowie die Element- bzw. Attributwerte (je ein Tripel pro Kante unter Verwendung von ElementIds):

type (source, target, value)

Beispiel: *literature*

source	target	value
1	2	-
25	26	-

author

source	target	value
3	-	Abiteboul
4	-	Buneman
5	-	Suciu
...		

year

source	target	value
2	-	2000
26	-	1998
...		

Abbildung von XML-QL auf SQL

// Query in XML-QL:

```
WHERE <article year=„1998">
            <author.name>$n</>
            <title>$t</>
          </>
          IN „www.books/bib.xml", $n LIKE „Suciu%", $t LIKE „%web%"
CONSTRUCT <result>$n $t</>
```

// zugehörige SQL-Query:

```
SELECT B.value, T.value                    // Name und Titel (Rückgabe)
FROM   article A, author B, name N, title T, year Y
WHERE  A.target = B.source                 // Kante article–author
  AND  A.target = T.source                 // Kante article–title
  AND  A.target = Y.source                 // Kante article–year
  AND  B.target = N.source                 // Kante author-name
  AND  N.value LIKE „Suciu%"
  AND  Y.value = „1998" AND T.value LIKE „%web%"
```


7.3 Erweiterung von XML-QL um Keyword-Suche

Bisher in XML-QL:
- weniger Wissen über die Struktur macht gezieltes Abfragen von Informationen schwieriger
- Query wird komplexer durch die verschiedenen Fälle: Elementname, Elementinhalt, Attributname, Attributwert

→ Erweiterung:
- Keyword-Suche über Element- und Attributnamen
- Keyword-Suche über (unstrukturierte) Elementinhalte

XML-QL-Erweiterung: Contains-Operator

```
CONTAINS(element_var, keyword, depth, expr)

    //suche nach „name“ als Subelement oder Attribut, bis Tiefe 3

    WHERE <article year=„1998“>
            <author></> ELEMENT_AS $n
            <title>$t</>
          </>
          IN „www.books/bib.xml“,
          CONTAINS ($n, „Suciu“, 3, any),
          $t LIKE „%web%“
    CONSTRUCT <result>$n $t</>

any = el_name OR attr_name OR el_content OR attr_value
```


Keyword-Index in relationaler DB

erweitertes invertiertes File:

keywordindex (keyword, elID, depth, location)

// location: element, attribut, content, value

```
<„literature“, elID 1, 0, tag>
<„book“,       elID 2, 1, tag>
<„year“,       elID 2, 1, attr>
...
<„author“,     elID 5, 2, tag>
<„Suciu“,      elID 5, 2, content>
...
```

Abbildung von Anfragen auf SQL unter
Verwendung von „keywordindex“ und (materialiserter) Views

Abbildung Keyword-Suche auf SQL

// Query XML-QL mit Stichwort-Suche:

```
WHERE <article year=„1998“>
            <author></> ELEMENT_AS $n
            <title>$t</> </>
          IN „www.books/bib.xml“,
          CONTAINS($n, „Suciu“, 3, any), $t LIKE „%web%“
CONSTRUCT <result>$n $t</>
```

// zugehörige SQL-Query:

```
SELECT B.value, T.value                    // Name und Titel (Rückgabe)
FROM   article A, author B, title T, year Y, keywordindex S
WHERE  S.keyword = „Suciu“                 //Keyword-Index
   AND S.depth < 4                         // Tiefenbeschränkung
   AND S.elID = B.source                   //Kante keywordindex-author
   AND B.source = A.target                 //Kante author-article
   AND A.target = T.source                 //Kante article–title
   AND A.target = Y.source                 //Kante article–year
   AND Y.value = „1998“ AND T.value LIKE „%web%“
UNION SELECT B.value, T.value FROM ... WHERE ... S.elID = T.source ...
UNION ...
```


7.4 XXL: Ranked Retrieval auf XML-Daten

XML-Datenbanktechnologie alleine erlaubt nur Boolesches Retrieval und ist ungeeignet für
große, heterogene XML-Datenkollektionen (Web, Intranets),
die kein einheitliches Strukturschema haben.

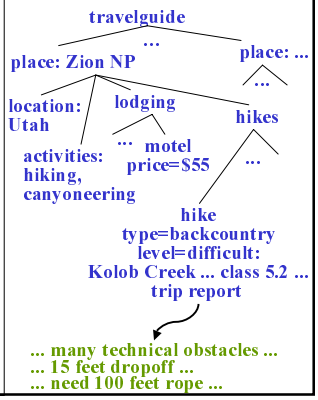
IR-Technologie alleine nutzt die durch XML-Elemente gegebenen semantisch reicheren Annotationen und kanonische Terminologie nicht aus.

→ **XXL (Flexible XML Search Language):**
Synergie von

- XML-QL-artigen Anfragesprachen
  (auf eine einfache Kernsprache reduziert) und
- Ranked Retrieval
  (mittels Ähnlichkeitsvergleichen auf Namen und Inhalten von Elementen und Attributen)

XML-Beispieldaten

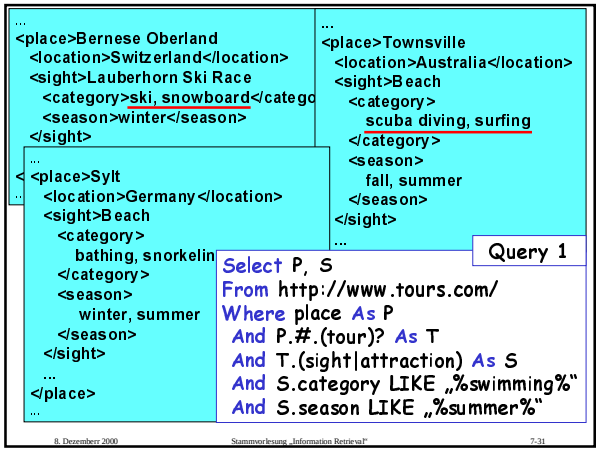
```
<travelguide>
  <place> Zion National Park
    <location> Utah </>
    <activities>
      hiking, canyoneering
    </activities>
    <lodging>
      <lodge price = $ 80-140>
        Zion Lodge ... </>
      <motel price = $55> ... </>
    </lodging>
    <hikes>
      <hike type=backcountry,
        level=difficult, href=... >
        Kolob Creek ... class 5.2 ...
      </hike>  ...
  </place>
  <place> Yosemite NP
    <location> California </>
    <activities> hiking, climbing
...
```

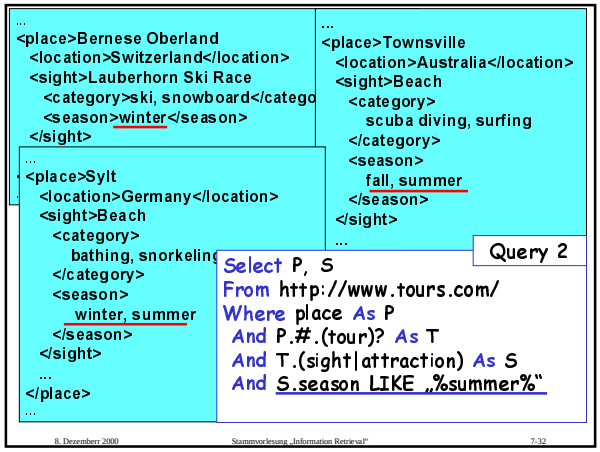


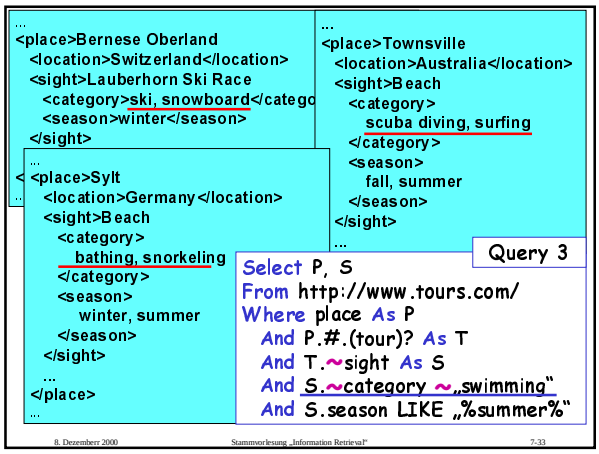
XML-Beispiel mit heterogenen Daten

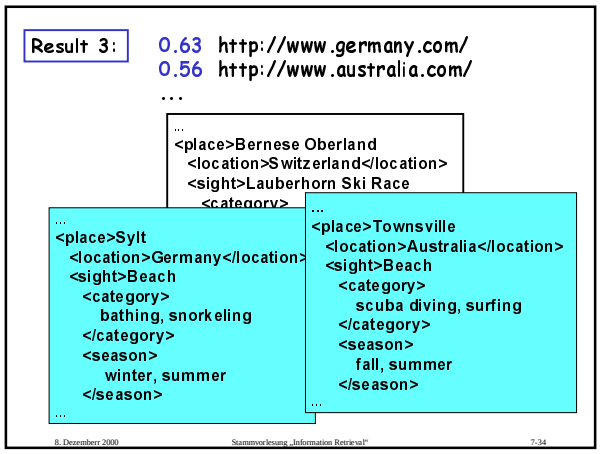
```
<travelguide>
  <place> Zion National Park
    <location> Utah </>
    <activities>
      hiking, canyoneering
    </activities>
    <lodging>
      <lodge price = $ 80-140>
        Zion Lodge ... </>
        price = $55> ... </>
    </lodging>
    <hikes>
      <hike type=backcountry,
        level=difficult>
        Kolob Creek ... class 5.2 ...
      </hike>  ...
  </place>
  <place> Yosemite NP
    <location> California </>
    <activities> hiking, climbing
...
```

```
<outdoors>
  <region> Zion NP
    <state> Utah </>
    <things-to-do>
      hiking, canyoneering
    </things-to-do>
    <campground
        fee = $15>
      ... </>
    <backcountry trips>
      <trip>
        Kolob Creek ...
        challenging hike ...
      </trip>  ...
  </region>
...
```









XXL: Where-Klausel

- logische Konjunktion regulärer Pfadausdrücke (And)

```
place.#.(tour)?.(sight|attraction).category=„swimming"
And place.#.(tour)?.(sight|attraction).season=„summer"
```

- reguläre Pfadausdrücke und Variable (+, ?, *, n, -n, # als Wildcard)

```
place.#.(tour)?.(sight|attraction).season
```

```
place.#.(tour)?.(sight|attraction).season As A
```

- elementare Bedingungen über Namen und Inhalt eines Elements oder Attributs (=, <>, LIKE, % als Wildcard...)

```
category LIKE „%swimming%"
```

8. Dezemberr 2000 Stammvorlesung „Information Retrieval" 7-35

XXL-Syntax

Struktur einer Anfrage:

Select-From-Where

Produktionsregeln für die Where-Klausel:

S	→ path | path **AND** path	
path	→ argument | path.path | (path' |' path) | (path)? | (path)+ | (path)* | (path)n | (path)-n | # | #n | #-n | path **AS** v | path.v.path	
argument	→ $object_{LABELEXPR}$ | $object_{VAR}$ | $object_{CONST}$ | uop argument | argument bop argument	
$object_{LABELEXPR}$	→ l | l.**NAME** | l.**CONTENT**	
l	→ LABELEXPR	
$object_{VAR}$	→ v | v.**NAME** | v.**CONTENT**	
v	→ VAR	
$object_{CONST}$	→ c | d **and** c | d **or** c	
c	→ CONST	
d	→ CONST	
uop	→ ~	
bop	→ = | ≠ | LIKE | …	

mit Terminalsymbolen:
$LABELEXPR \cup VAR \cup CONST \cup OP$

8. Dezemberr 2000 Stammvorlesung „Information Retrieval" 7-36

Vereinfachte XXL-Semantik (1)

Ein **Element** ist ein 5-Tupel der Form (oid, label, content, elements, attributes) mit einem Bezeichner oid, einem String label, einem String content, einer Liste elements von weiteren Elementen, die auch leer sein kann, und einer Liste attributes von Attributen.
Ein **Dokument** ist ein Element, das in keinem anderen Element enthalten ist.
Für ein Element x bezeichnen x.oid, x.label, x.content, x.elements die entsprechenden Komponenten; x.elements[i] bezeichnet das i-te Element.
Ein **Datengraph** ist der einem Dokument x entsprechende Graph G=(V,E) mit Elementen als Knotenmenge V und einer Kantenmenge E, die eine Kante von a nach b enthält, wenn b in a.elements ist. Knoten sind mit den Labels der Elemente markiert. Die Reihenfolge in a.elements wird im Graphen nicht mehr berücksichtigt. Ein Pfad p im Graph entspricht einer Label-Sequenz.

Notation:
Für ein XML-Dokument x mit Graph G=(V,E) bezeichnen
- LABEL $\subseteq \Sigma^*$ die Menge der in x vorkommenden Labels,
- PATH $\subseteq V^+$ die Menge der Pfade in G (der Länge k für k Knoten und k-1 Kanten für k=1, 2, ...),
- VAR eine Menge von Knotenvariablen.

Vereinfachte XXL-Semantik (2)

LABELEXPR ist die Menge $(\Sigma \cup \{\%\})^+$ mit dem Wildcard-Zeichen %.
PATHEXPR, ist die Menge der **regulären Pfadausdrücke**, d.h. die kleinste Menge mit den folgenden Eigenschaften:
- Jedes Wort aus LABELEXPR ist ein Pfadausdruck.
- Jede Variable A $\in$ VAR ist ein Pfadausdruck (für Pfad der Länge 1).
- Falls S und T Pfadausdrücke sind, dann sind auch
  ST (Konkatenation), S|T (Vereinigung), (S) (Klammerung),
  (S)? (optionales Vorkommen), (S)+ (mehrfaches Vorkommen),
  (S)* (optional mehrfaches Vorkommen) und # (Abk. für (%)*)
  Pfadausdrücke.

Eine **Variablenbindung** zu einem gegebenen Datengraphen G=(V,E) ist eine Abbildung ν: VAR → V. ν[A] bezeichnet den Wert der Variablen A.

Vereinfachte XXL-Semantik (3)

Ein Pfad p eines Datengraphen G **erfüllt** einen Pfadausdruck E bei Variablenbindung ν, wenn folgendes gilt:
- Falls E ein Labelausdruck ist, dann muß p Länge 1 haben und dem Stringmuster E $\in (\Sigma \cup \{\%\})^+$ entsprechen.
- Falls E eine Variable A ist, dann muß p Länge 1 haben und ν[A]=first(p) sein.
- Falls E die Form ST hat, dann muß p zerlegbar sein in pre(p) und suf(p) mit p=pre(p)suf(p), |pre(p)| ≥1, |suf(p)|) ≥1, so daß pre(p) S erfüllt und suf(p) T erfüllt.
- Falls E die Form S|T hat, dann muß p S oder T erfüllen.
- Falls E die Form (S) hat, dann muß p S erfüllen.
- Falls E die Form (S)? hat, dann muß p leer sein oder S erfüllen.
- Falls E die Form (S)+ hat, dann muß p zerlegbar sein in pre(p) und suf(p) mit p=pre(p)suf(p), |pre(p)| ≥1, |suf(p)|) ≥1, so daß pre(p) S erfüllt und suf(p) (S)+ erfüllt.
- Falls E die Form (S)* hat, dann muß p leer sein oder (S)+ erfüllen.
- Falls E die Form # hat, erfüllt p E.

Vereinfachte XXL-Semantik (4)

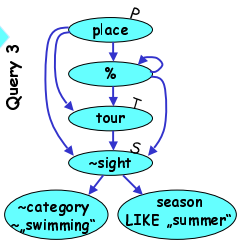
Eine **elementare Query** Q auf einem Datengraphen G=(V,E) ist von der Form 1) E As A oder 2) [E As] A op b oder 3) [E As] A op B mit
E $\in$ PATHEXPR, op $\in$ {<, >, ≤, ≥, =, ≠, LIKE}, Variablen A, B und Konst. b.
Eine **Query** Q auf G=(V,E) ist eine Konjunktion von elementaren Queries.

Die **Semantik einer elementaren Query** Q auf G bei Variablenbelegung ν ist:
$\mu_\nu[Q, G] = \{p \in V^+ \mid$ p erfüllt E und es gilt
last(p).content op ν[A] op b bzw. last(p).content op ν[A] op ν[B].
Die **Semantik einer Query** Q = Q1 & Q2 & ... & Qm mit elementaren Queries Q1, ..., Qm auf G ist:
$\mu[Q, G] = \{$Teilgraphen G' von G | G' ist die Vereinigung von Pfaden p1, ..., pm $\in V^+$ und es gibt eine Variablenbelegung ν, so daß für alle i=1, ..., m gilt: pi $\in \mu_\nu[Q, G]\}$.

XXL: Pfadausdruck → NEA

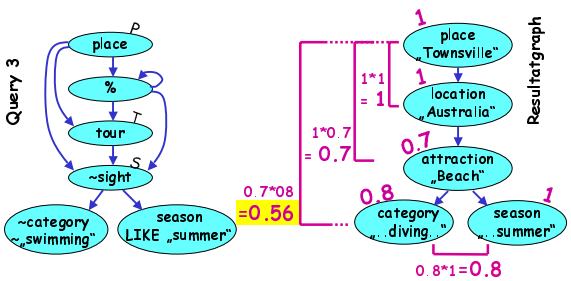
Where-Klausel einer Query
- Ähnlichkeitsoperator ~

```
Query 3
Select P, S
From http://www.tours.com/
Where place As P
  And P.#.(tour)? As T
  And T.~sight As S
  And S.~category~„swimming"
  And S.season LIKE „%summer%"
```



XXL: Ähnlichkeitsoperator ~ (1)

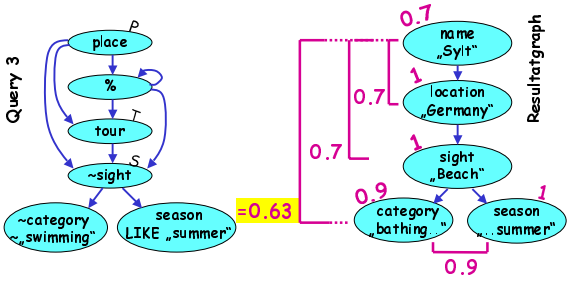
Where-Klausel einer Query
- Ähnlichkeitsoperator ~
- Regeln zur Relevanzbewertung von Knoten und Pfaden



XXL: Ähnlichkeitsoperator ~ (2)

Where-Klausel einer Query

- Ähnlichkeitsoperator ~
- Regeln zur Relevanzbewertung von Knoten und Pfaden



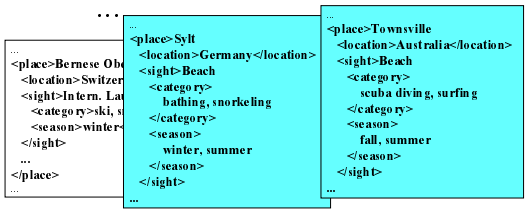
XXL: Ähnlichkeitsoperator ~ (3)

Query 3:

```
Where place As P
  And P.#.(tour)? As T And T.~sight As S
  And S.~category~„swimming"
  And S.season LIKE „%summer%"
```

Result 3:

0.63 http://www.germany.com/
0.56 http://www.australia.com/
...



XXL-Semantik mit Ähnlichkeitsoperator ~ (1)

Relevanz-Score für primitive Bedingungen:

- Für Elementinhaltsvergleiche basiert der Score auf der tf*idf-Formel.
- Für Element- und Attributnamensvergleiche basiert der Score wahlweise auf der
  - Levenshtein-String-Ähnlichkeit oder
  - einer „semantischen" Ähnlichkeitsfunktion auf einem Ontologiebaum für Elementnamen
- Für Attributwertevergleiche ist der Score typspezifisch (z.B. basierend auf Intervallen bei Geldbeträgen oder Datumsangaben oder berechneten Distanzen, Fahrzeiten usw. bei Ortsangaben)

XXL-Semantik mit Ähnlichkeitsoperator ~ (2)

Relevanz-Score für Pfade:

- Für Pfade p1 and p2 mit Scores π1 and π2 bzgl. Bedingungen c1 und c2 ist der Score von p1.p2 bzgl. "c1.c2" π1*π2.
- Für Pfad p mit Score π bzgl. c ist der Score von p bzgl. "c?" π.
- Für Pfade p1, p2, ..., pm mit Scores π1, π2, ..., πm bzgl. c ist der Score von p1.p2. ... .pm bzgl. "c+", "c*", "cn" und "c-n" π1*π2*...*πm bzw.
  π1*π2*...*πm falls p1.p2. ... .pm nicht leer ist, 1 sonst bzw.
  π1*π2*...*πm falls n=m, 0 sonst bzw.
  π1*π2*...*πm falls m≤n, 0 sonst.
- Für Pfad p und Bedingung "%" (z.B. "(%)*" bzw. "#") ist der Score 1.

Relevanz-Score für Teilgraphen:

- Für Teilgraphen g1 and g2 mit Scores π1 und π2 bzgl. c1 und c2 ist der Score von g1 ∪ g2 bzgl. "c1 | c2" π1 + π2 – π1*π2 und der Score von g1 ∪ g2 bzgl. "c1 & c2" π1*π2.
- Für Teilgraphen g1 and g2 mit Scores π1 und π2 bzgl. c1 and c2 ist der Score von g1.g2 bzgl. "c1.c2" π1*π2.

XXL-Semantik mit Ähnlichkeitsoperator ~ (3)

Für einen Ontologiebaum (V, E) mit gewichteten Kanten ist die Distanz zwischen zwei Knoten, *dist*: $V \times V \rightarrow$ [0..1], wie folgt definiert:

- Für Knoten v_1 und v_2 auf demselben Wurzel-Blatt-Pfad ist $dist\ (v_1, v_2)$ die Anzahl der Kanten zwischen v_1 und v_2.
- Für Geschwisterknoten v_1, v_2 mit Vater p ist
  $siblingdist\ (v_1, v_2) = |weight\ (p, v_1) - weight\ (p, v_2)\ | / |children\ (p)|$
- Für beliebige Knoten v_1, v_2 mit dem kleinsten gemeinsamen Vorfahren p, der Kinder p_1 und p_2 mit Pfaden $p.p_1 \ldots v_1$ und $p.p_2 \ldots v_2$ hat
  $dist\ (v_1, v_2) = (\ length(p_1, v_1) + length\ (p_2, v_2) + siblingdist\ (p_1, p_2) + 1\ )$
  $/\ maxdist\ (v_1, v_2)$,
  wobei $maxdist\ (v_1, v_2) = length\ (r, v_1) + length\ (r, v_2)$
  ein Normierungsfaktor und r die Wurzel des Baums ist.

Die Ähnlichkeit zweier Knoten ist dann: $sim\ (v_1, v_2) = 1 - dist\ (v_1, v_2)$.

Implementierung mit Oracle8i interMedia

✪ Abbildung von XML-Dokumenten auf relationale Tabellen:

Documents (URL, root_oid) Attributes (oid, name, value)
Elements (oid, name, content) Edges (oid, parent_oid)

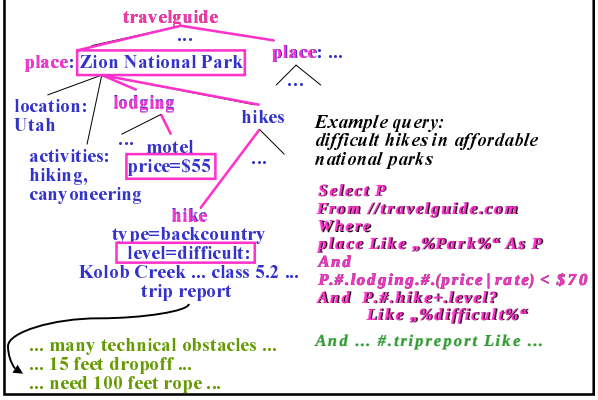
```
Select P, S From http://www.tours.com/
Where place As P And P.#.(tour)? As T And T.~sight As S
  And S.~category~„swimming" And S.season=„summer"
```

✪ Abbildung von XXL-Anfragen auf SQL und Oracle8i interMedia
→ Testen elementarer Bedingungen + Aufbau des Resultatgraphs

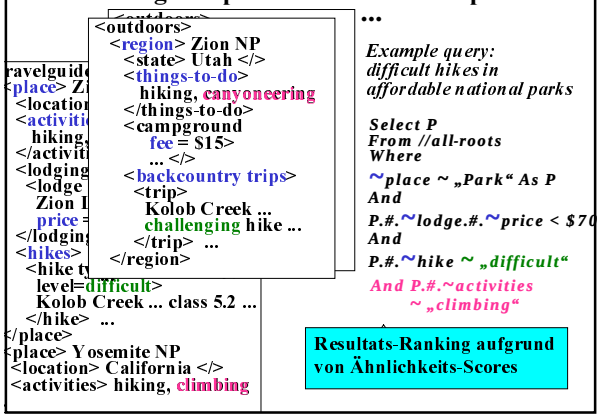
Anhängen an Resultatgraph

```
Select e, SCORE(1)+SCORE(2)-SCORE(1)*SCORE(2)
From Elements e, Edges v
Where CONTAINS (e.name, SYN(category), 1)>0
  And CONTAINS (e.content, „?swimming", 2)>0
  And e.oid = v.oid
  And v.parent_oid=a current leaf in resultgraph
```

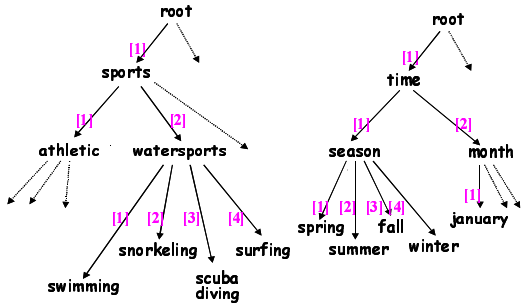
XXL-Anfragebeispiel



XXL-Anfragebeispiel mit Ähnlichkeitsoperator ~



Beispiel für Ontologie-basierte Ähnlichkeit



sim (swimming, snorkeling) = $1 - ((1/4 + 1)/6) \approx 0.792$
sim (swimming, surfing)= $1 - ((3/4 + 1)/6) \approx 0.709$
sim (season, time) = $1 - 1/3 \approx 0.666$