

Datenmanagement jenseits von Relationen

Kapitel 2.3: Relationale Speicherung von XML Daten
– Der XPath-Accelerator

Lernziele für heute ...

- Was ist das EDGE Model
 - Warum ist es nicht ausreichend
- Wie funktioniert der XPath-Accelerator
 - R-Bäume
 - Abbildung der XPath Achsen auf 2D Koordinaten
 - Relationale Umsetzung



Beispiel für relationale Speicherung

Dokument

```
<rezept>
  <zutaten id="x1">
    <zutat>Ei</zutat>
    <zutat>Mehl</zutat>
  </zutaten>
<expertise/>
<zutaten id="x2">
  <zutat>Salz</zutat>
</zutaten>
</rezept>
```

Mögliche relationale Darstellung

Source	Name	VString	Target
1	rezept		x1
x1	zutaten		2
x1	zutaten		3
2	zutat	Ei	
3	zutat	Mehl	
1	rezept		4
4	expertise		
1	rezept		x2
...
5	zutat	Salz	

- Beispiel für (relationale) materialisierte Sicht auf das XML Dokument.
- Relationale Darstellung ist Dokumenttyp-unabhängig.

EDGE-
Modell

Beispiel für relationale Speicherung

Dokument

```
<rezept>
  <zutaten id="x1">
    <zutat>Ei</zutat>
    <zutat>Mehl</zutat>
  </zutaten>
  <expertise/>
  <zutaten id="x2">
    <zutat>Salz</zutat>
  </zutaten>
</rezept>
```

Mögliche relationale Darstellung

Source	Name
1	rezept
x1	zutaten
4	expertise
X2	zutaten
2	zutat
3	zutat
5	zutat

Source	Target
1	x1
x1	2
x1	3
1	4
1	x2
...	...

Source	VString
2	Ei
3	Mehl
5	Salz

- Beispiel für (relationale) materialisierte Sicht auf das XML Dokument.
- Relationale Darstellung ist Dokumenttyp-unabhängig.

EDGE-
Modell

Beispiel für relationale Speicherung

Dokument

```
<rezept>
  <zutaten id="x1">
    <zutat>Ei</zutat>
    <zutat>Mehl</zutat>
  </zutaten>
  <expertise/>
  <zutaten id="x2">
    <zutat>Salz</zutat>
  </zutaten>
</rezept>
```

Mögliche relationale Darstellung

Source	Name
1	rezept
x1	zutaten
4	expertise
X2	zutaten
2	zutat
3	zutat
5	zutat

Source	Target
1	x1
x1	2
x1	3
1	4
1	x2
...	...

Source	VString
2	Ei
3	Mehl
5	Salz

Warum ist das EDGE-Modell
nicht zufriedenstellend?

EDGE-
Modell

Evaluierung von XPath-Ausdrücken mit relationalen Datenbanken

- Ziel: Effiziente evaluierung von XPath-Ausdrücken innerhalb relationaler Datenbanken
 - Erste Ansätze mittels Data Guide
 - Noch offen: **XPath Achsen**
- Problem: Naive Lösungen wie EDGE-Modell
 - offensichtlich sehr teuer, hohe Join-Kosten.
- Teilziele
 - Abbildung von XPath Achsen nach SQL
 - Zugehörige Abbildung von XML auf Relationen
 - Leverage relationaler Datenbank-Technologie.

Gliederung im Folgenden

- Räumliche Anfragen mit R-Bäumen
- XPath Achsen im Dokument und ihre Repräsentation,
- Optimierungen,
- Umsetzung in SQL.

Accelerating XPath Evaluation in Any RDBMS

TORSTEN GRUST
University of Konstanz
and
MAURICE VAN KEULEN
University of Twente
and
JENS TEUBNER
University of Konstanz

This article is a proposal for a database index structure, the *XPath accelerator*, that has been specifically designed to support the evaluation of XPath path expressions. As such, the index is capable to support *all* XPath axes (including *ancestor*, *following*, *preceding-sibling*, *descendant-or-self*, *etc.*). This feature lets the index stand out among related work on XML indexing structures which had a focus on the *child* and *descendant* axes only. The index has been designed with a close eye on the XPath semantics as well as the desire to engineer its internals so that it can be supported well by *existing* relational database query processing technology: the index (a) permits set-oriented (or, rather, sequence-oriented) path evaluation, and (b) can be implemented and queried using well-established relational index structures, notably B-trees and R-trees.

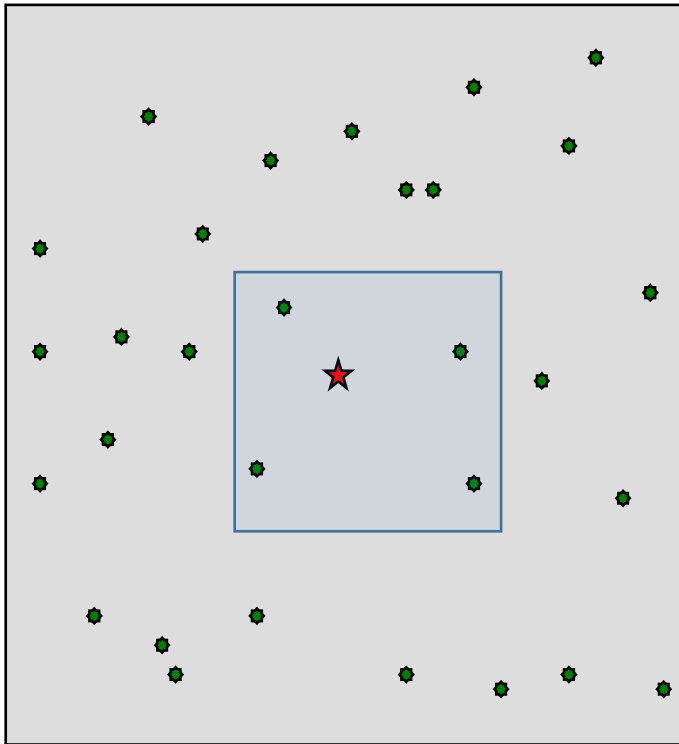
We discuss the implementation of the XPath accelerator on top of different database backends and show that the index performs well on all levels of the memory hierarchy, including disk-based and main-memory based database systems.

Categories and Subject Descriptors: H.2.4 [Database Management]: Systems—*relational databases; query processing*; E.1.0 [Data Structures]: Trees

General Terms: Performance, Theory

Additional Key Words and Phrases: Main-memory databases, XML, XML indexing, XPath

Räumliche Anfragen – Motivation



Datenraum

■ Anwendung:

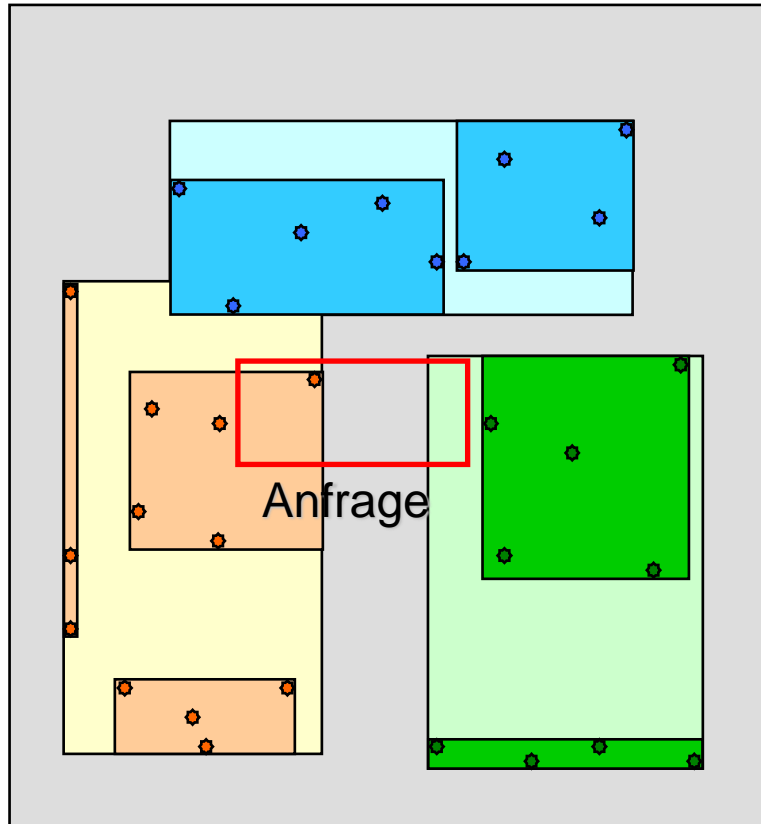
- Grüne Punkte: Bars, die Ihr bevorzugtes Bier ausschenken.
- Stern: Ihr Standort.
- Punkte enthalten in Relation Bar.

x	y	Name
8	50	Uno
9	48	Plus
7	52	Joker
9	52	Adonis

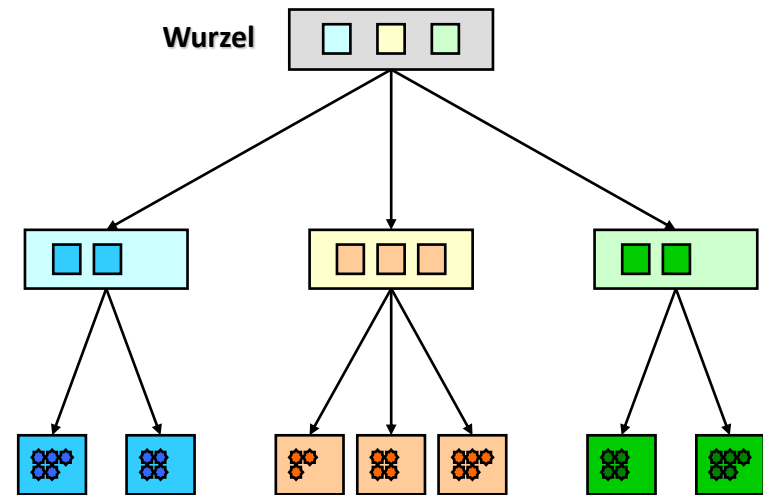
Offensichtliche Lösung:
Relation scannen,
Abstand jedes Tupels
berechnen.

- Welche Bars sind in der Nähe?

R-Baum



Datenraum

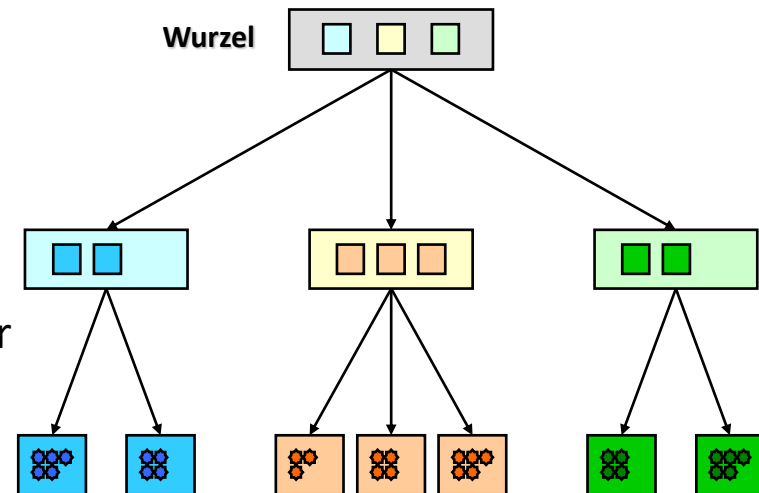


R-Baum

Kosten der Anfrageevaluierung i. d. R. abhängig von Fenstergröße.

R-Baum – Take away

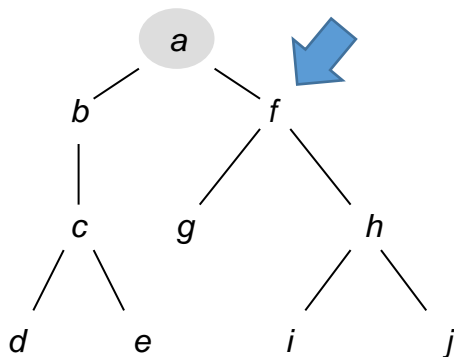
- Index in vielen DBMS verfügbar
- Funktioniert besonders gut für
 - Fensteranfragen mit kleinem Fenster
 - Datensätze mit geringer Dimensionalität (2D)



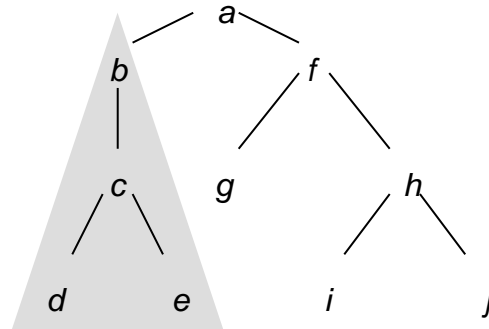
R-Baum

XPath Achsen (1) – Wdh.

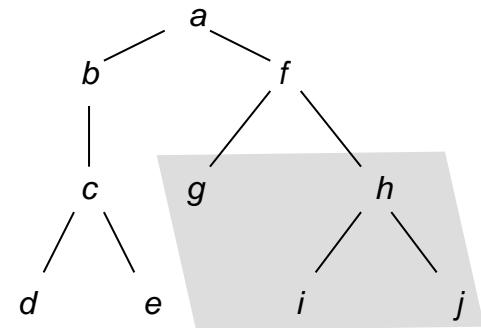
- Teilmenge von Knoten eines Dokuments, aus Perspektive eines ausgezeichneten Knotens.
- Vier „wesentliche“ Achsen: ancestor, preceding, descendant, following.



(a)



(b)



(c)

XPath Semantik: Knoten, die vom **Knoten f** aus erreichbar sind, entlang der folgenden Achsen:

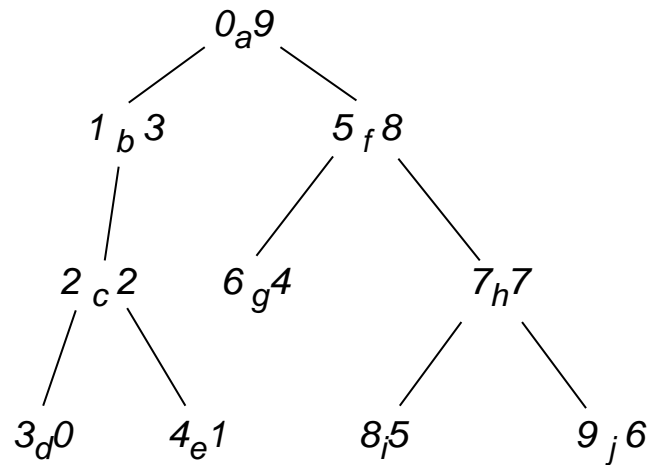
(a) ancestor, (b) preceding, (c) descendant.

XPath Achsen (2)

- Beobachtung: „Wesentliche“ Achsen partitionieren Menge der Knoten, d. h.:
- Die Vereinigung der 4 Achsen und v:
 - $v/\text{descendant} \cup v/\text{ancestor} \cup v/\text{following} \cup v/\text{preceding} \cup \{v\}$
 - enthält jeden Knoten des Dokuments genau einmal.
- Wie hilft das jetzt?

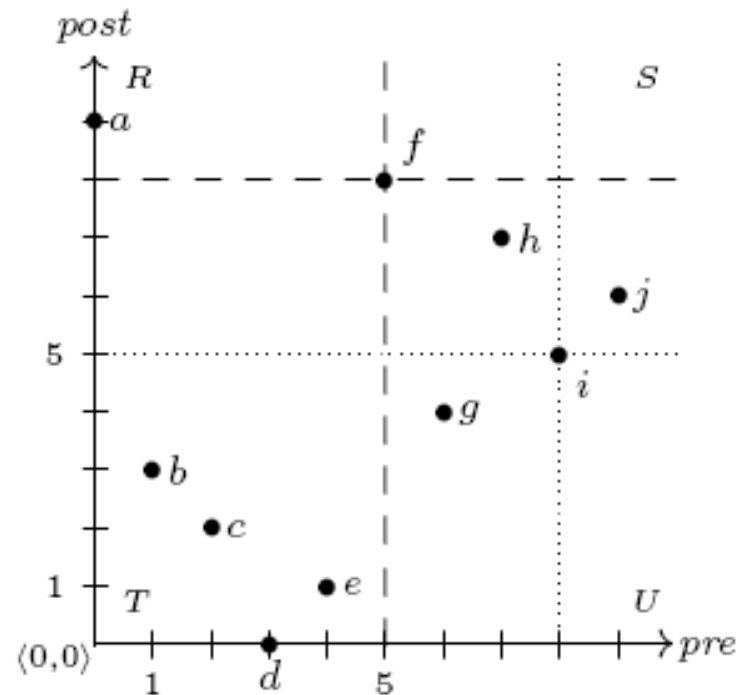
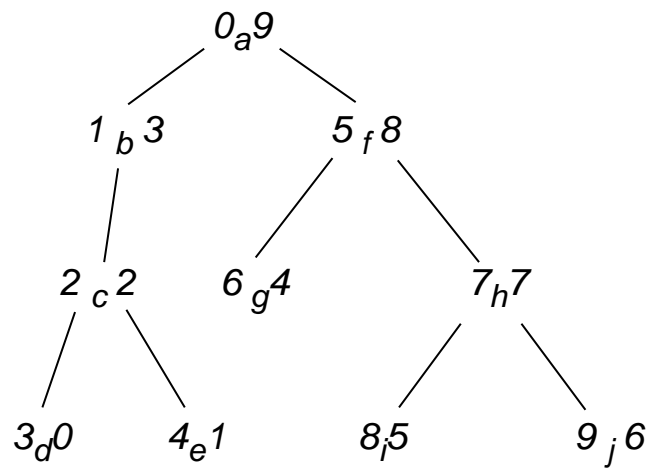
Repräsentation von Regionen im Dokument (1)

- Preorder – Tiefensuche.
- Postorder – jeder Knoten vor seinem Vater und vor seinem rechten Nachbarn.



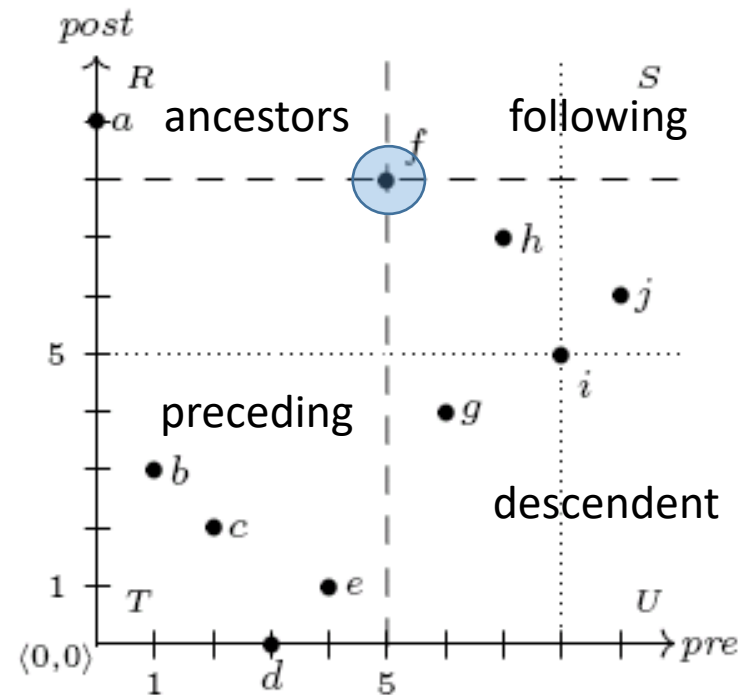
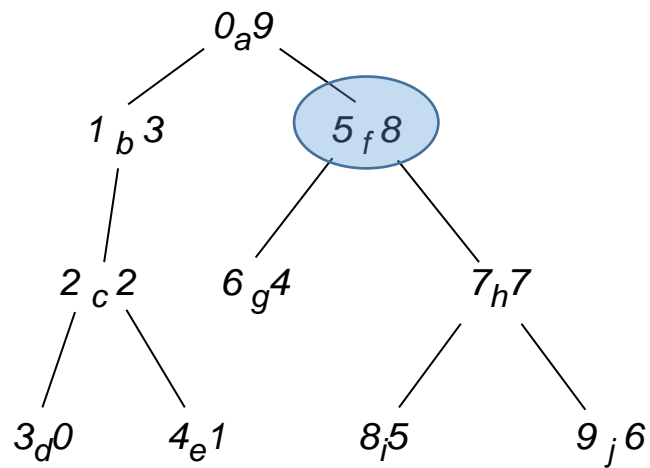
Repräsentation von Regionen im Dokument (2)

- Preorder
- Postorder



Repräsentation von Regionen im Dokument (2)

- Preorder
- Postorder



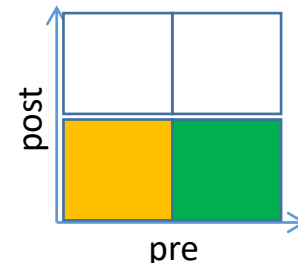
Repräsentation von Regionen im Dokument (3)

- Alle Achsen als einer der vier Quadranten mit Zentrum des Anfrageknotens (v) darstellbar
- Bsp.: v' ist Descendant von v
 $\Leftrightarrow \text{pre}(v) < \text{pre}(v') \wedge \text{post}(v') < \text{post}(v)$
- Zusammenhang zwischen o. g. Partitionierung und pre-/post-Ebene.

Repräsentation der Knoten als Vektoren

preorder von parent

- Deskriptor von v :
 $\text{desc}(v) = \langle \text{pre}(v), \text{post}(v), \text{par}(v), \text{kind}(v), \text{name}(v) \rangle$
- Vektor auch zur Beschreibung der Anfrage.
Absolute Werte oder Intervalle angebbbar.
- Genauer: Mit Deskriptoren
sind Location Steps einfach abbildbar, z. B.
 - $\text{window}(\text{preceding}, v)$
 $= \langle [0, \text{pre}(v), [0, \text{post}(v)), *, *, * \rangle$
 - $\text{window}(\text{descendant}, v)$
 $= \langle (\text{pre}(v), \text{infinity}), [0, \text{post}(v)), \dots \rangle$
 - $\text{window}(\text{child}, v) = \langle *, *, \text{pre}(v), *, * \rangle$
 - Elternknoten von v ?



Repräsentation der Knoten als Vektoren (2)

- Vektor auch zur Beschreibung der Anfrage. Absolute Werte oder Intervalle angebbbar.
- Genauer: Mit Deskriptoren sind Location Steps einfach abbildbar, z. B.
 - `window(preceding::text() , v)`
 $= \langle [0, \text{pre}(v)), [0, \text{post}(v)), *, \text{text}, * \rangle$

Obere/untere
Grenze pre
wert

Obere/untere
Grenze post
wert

Zusatzeigensch-
aften z.B. nur
Text-Knoten

- `window(preceding::text() , f)`
 $= \langle [0, 5), [0, 8), *, \text{text}, * \rangle$

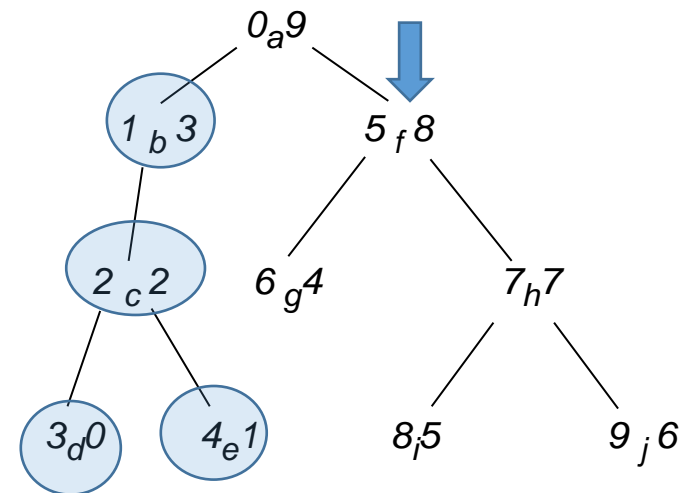


Abbildung der Achsen

Axis α	pre	$post$	par	$kind$	$name$
child	$\langle (pre(v), \infty)$	$, [0, post(v))$	$, pre(v)$	$, elem$	$, * \rangle$
descendant	$\langle (pre(v), \infty)$	$, [0, post(v))$	$, *$	$, elem$	$, * \rangle$
descendant-or-self	$\langle [pre(v), \infty)$	$, [0, post(v)]$	$, *$	$, elem$	$, * \rangle$
parent	$\langle [par(v), par(v)]$	$, (post(v), \infty)$	$, *$	$, elem$	$, * \rangle$
ancestor	$\langle [0, pre(v))$	$, (post(v), \infty)$	$, *$	$, elem$	$, * \rangle$
ancestor-or-self	$\langle [0, pre(v)]$	$, [post(v), \infty)$	$, *$	$, elem$	$, * \rangle$
following	$\langle (pre(v), \infty)$	$, (post(v), \infty)$	$, *$	$, elem$	$, * \rangle$
preceding	$\langle [0, pre(v))$	$, [0, post(v))$	$, *$	$, elem$	$, * \rangle$
following-sibling	$\langle (pre(v), \infty)$	$, (post(v), \infty)$	$, par(v)$	$, elem$	$, * \rangle$
preceding-sibling	$\langle [0, pre(v))$	$, [0, post(v))$	$, par(v)$	$, elem$	$, * \rangle$
attribute	$\langle (pre(v), \infty)$	$, [0, post(v))$	$, pre(v)$	$, attr$	$, * \rangle$

Repräsentation der Knoten als Vektoren (3)

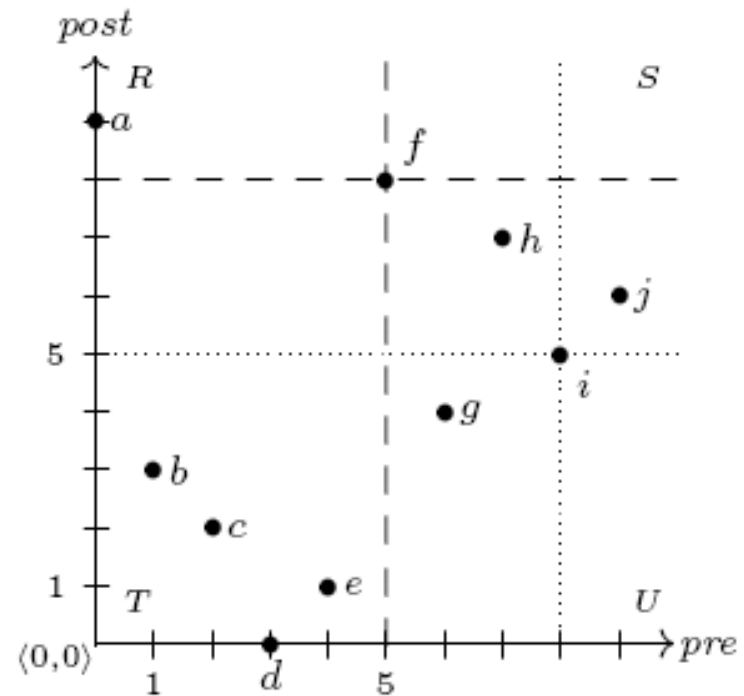
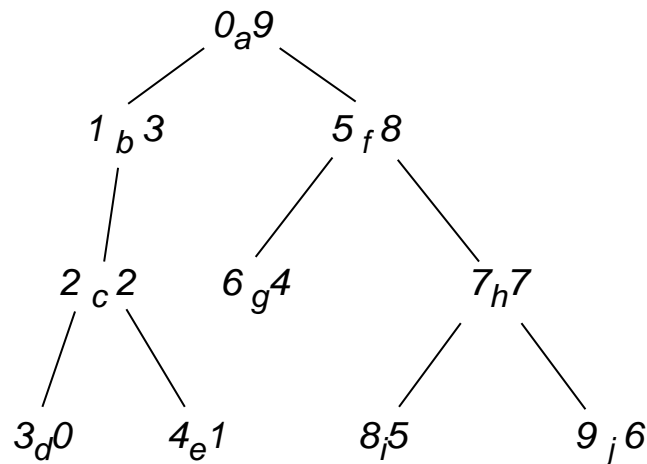
z. B. elem, attr, text

- $\text{descr}(v)$
= $\langle \text{pre}(v), \text{post}(v), \text{par}(v), \text{kind}(v), \text{type}(v) \rangle$
 - $\text{descr}(v)$ = Deskriptor von v ;
 - ‚name‘ ist leer,
wenn es sich um Text-Knoten handelt.
- Beispiel für Knoten f aus Bsp Folie 18
 - $\text{descr}(F) = \langle 5, 8, 0, \text{elem}, \text{rezept} \rangle$

<u>Pre</u>	Post	Parent	Kind	Type
5	8	0	elem	rezept

Zusammenfassung bis hierhin

- Auswertung **eines** XPath Location Steps:
Im wesentlichen Zugriff auf entsprechenden Ausschnitt
der pre-/post-Ebene.



Datenmanagement jenseits von Relationen

Kapitel 2.3: Relationale Speicherung von XML Daten
– Optimierungen des Xpath-Accelerators

Optimierungen

Übersicht:

- Verkleinerung des Fensters der pre-/post-Ebene, auf das man zugreifen muss.
- Zugriff ersetzen durch Zugriff mit Einschränkung der pre- oder der post-Achse.
- Ausnutzung von Symmetrien für die Optimierung der Auswertung.
- Auswertung von Pfadausdrücken für mehrere Knoten – Vermeidung mehrfacher Zugriffe auf den gleichen Knoten.

Verkleinerung des Fensters (1)

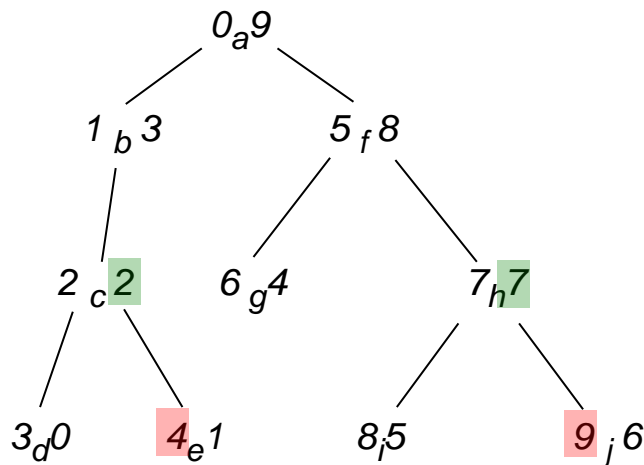
- Descendant Achse: Suche alle descendants von v

```
window(descendant, v)  
= <(pre(v), infinity), [0, post(v)), ...>
```

- Problem: Datenraumgrenzen im Intervall
 - oberer Preorder Wert: **infinity**
 - Unterer postorder Wert: **0**
 - Problematisch für R-Baum index

Verkleinerung des Fensters (1)

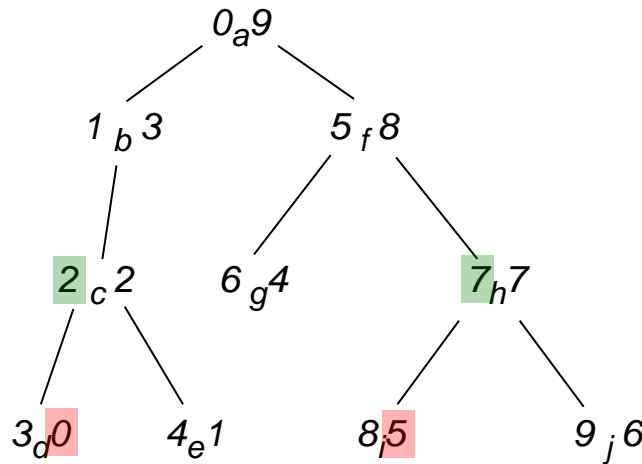
- Descendant Achse: Suche alle descendants von v
- Idee: Obere Schranke für **Preorder**-Nummer des Descendants.
 - Es gilt: $\text{pre}(v') \leq \text{post}(v) + \text{height}(t)$
 - v' ist Descendant von v .
 - $\text{height}(t)$ ist Höhe des Baums. (Im Beispiel ist $\text{height}(t)=3$.)
 - Kann man zeigen.



$\text{window}(\text{descendant}, v)$
 $= \langle (\text{pre}(v), \text{infinity}), [0, \text{post}(v)], \dots \rangle$

Verkleinerung des Fensters (2)

- Es gilt bereits: $\text{pre}(v') \leq \text{post}(v) + \text{height}(t)$
- Analog für **Postorder**:
 $\text{post}(v'') \geq \text{pre}(v) - \text{height}(t)$
(v'' – linkstes Blatt unter v = Knoten mit kleinsten pre Wert)

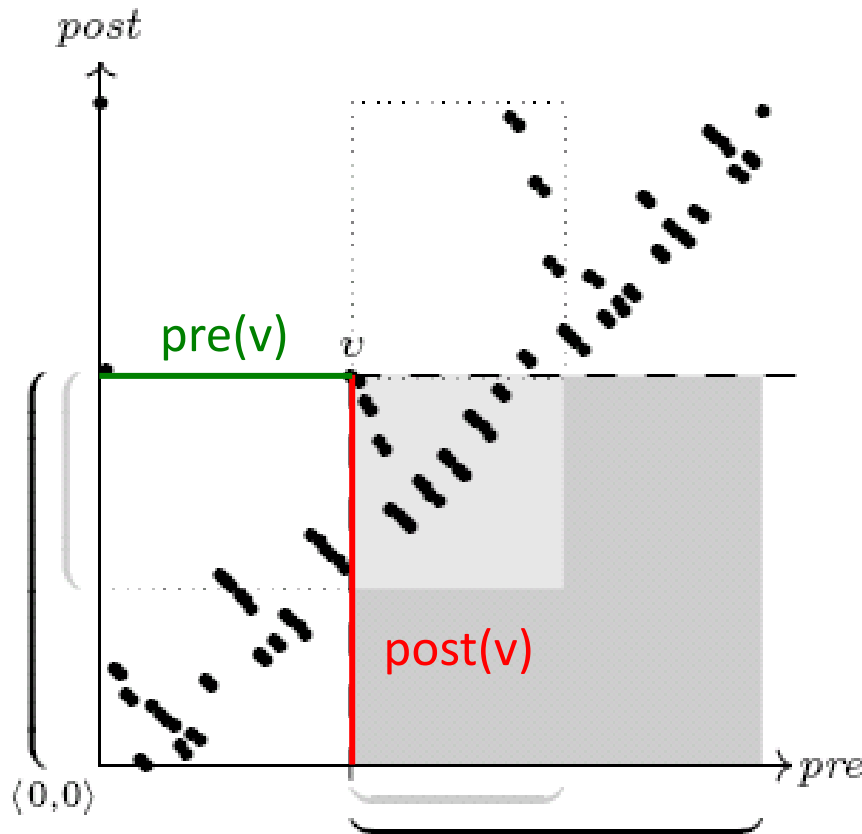


$\text{window}(\text{descendant}, v)$
 $= \langle (\text{pre}(v), \text{infinity}), [\mathbf{0}, \text{post}(v)], \dots \rangle$

Verkleinerung des Fensters (3)

- Es gilt: $\text{pre}(v') \leq \text{post}(v) + \text{height}(t)$
 - Analog:
 $\text{post}(v'') \geq \text{pre}(v) - \text{height}(t)$
(v'' – linkstes Blatt unter v)
- Fenster, das man für Descendant-Zugriff inspizieren muss, verkleinert sich:
- Bis jetzt:
 $\text{window}(\text{descendant}, v) = \langle \text{pre}(v), \infty, [0, \text{post}(v)), \dots \rangle$
- Ergebnis der Optimierung:
 $\text{window}(\text{descendant}, v) = \langle \text{pre}(v), \text{post}(v) + \text{height}(t), [\text{pre}(v) - \text{height}(t), \text{post}(v)), *, \text{elem}, * \rangle$

Verkleinerung des Fensters (4)

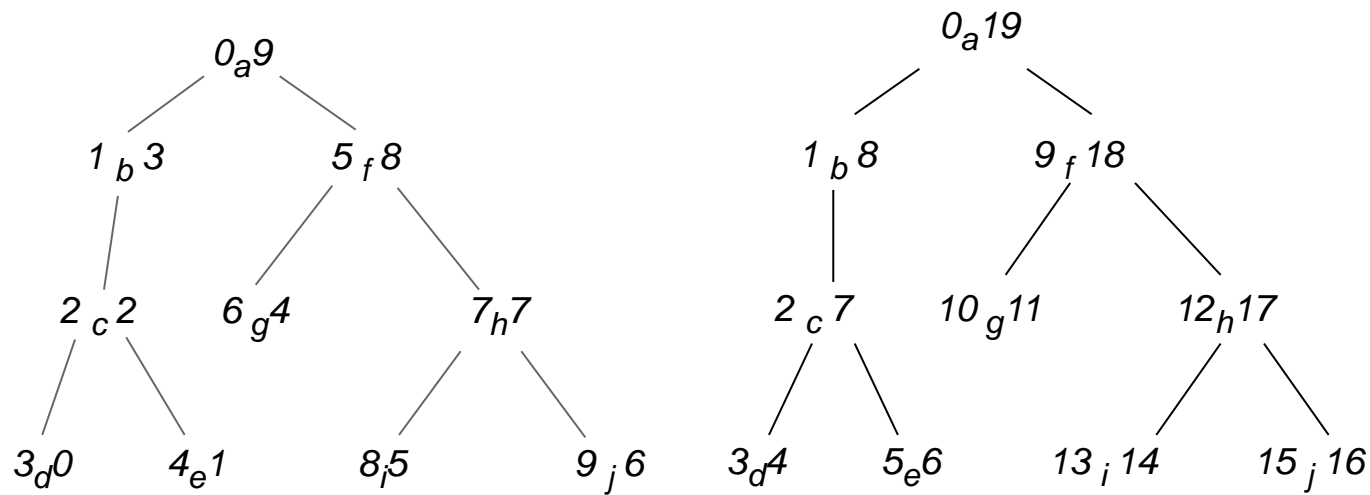


Bis jetzt:
 $\text{window}(\text{descendant}, v) =$
 $\langle (\text{pre}(v), \infty), [0, \text{post}(v)), \dots \rangle$

$\text{window}(\text{descendant}, v) =$
 $\langle (\text{pre}(v), \text{post}(v) + \text{height}(t)],$
 $[\text{pre}(v) - \text{height}(t), \text{post}(v)), *, \text{elem}, * \rangle$

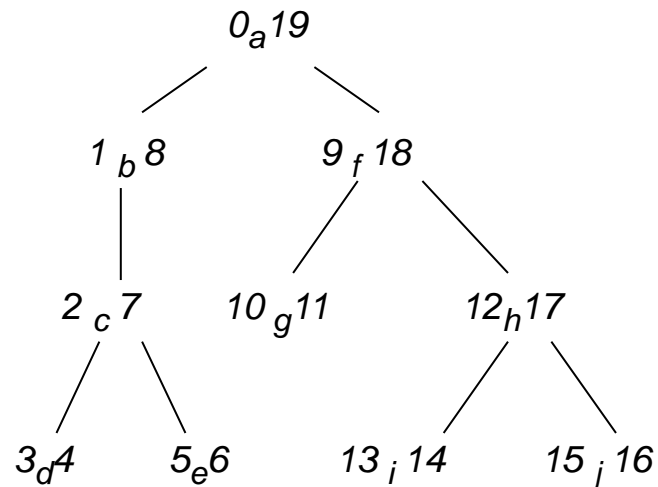
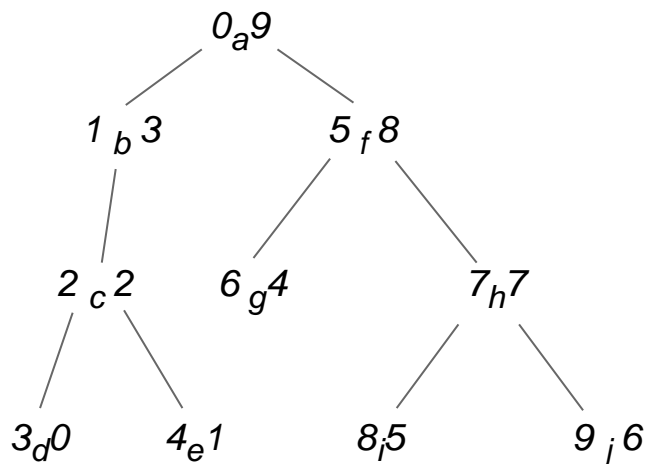
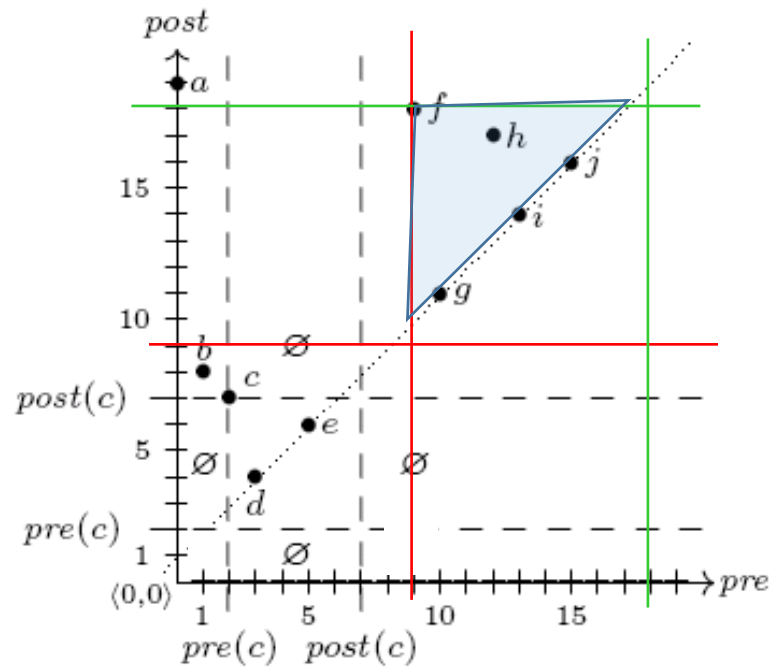
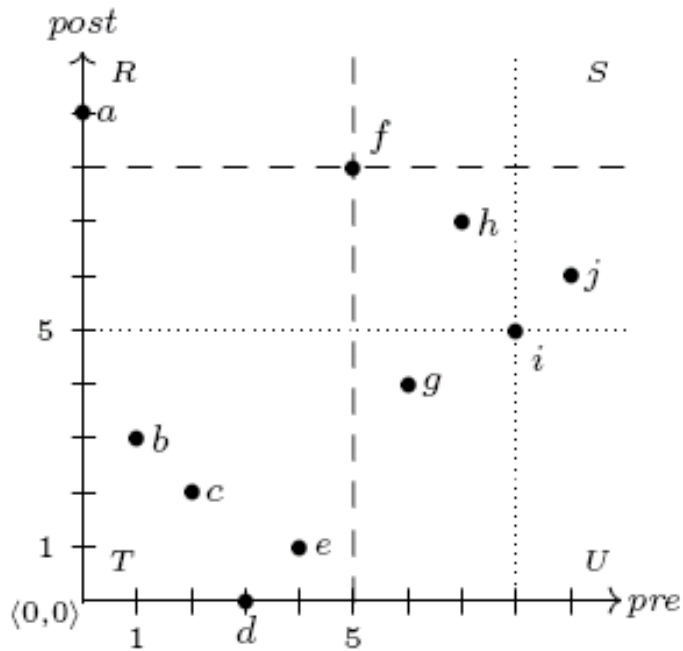
Zugriff mit nur **einer** Achse (1)

- Auswertung der Prädikate bisher:
Range Query in zwei Dimensionen (pre, post).
- Jetzt: Modifikation von pre und post.



- $\text{pre}(c) < \text{pre}(v) < \text{post}(c)$
- $\text{pre}(c) < \text{post}(v) < \text{post}(c)$
- Annotation durch linke Tiefensuche: erster und letzter Besuch eines Knotens

Zugriff mit nur **einer** Achse (2)



Zugriff mit Einschränkung nur einer Achse (3)

- Maßnahme **nur** für descendants-Suche.

Datenmanagement jenseits von Relationen

Kapitel 2.3: Relationale Speicherung von XML Daten
– Umsetzung in SQL

Umsetzung in SQL (1)

- Eine zentrale Relation mit Schema
(pre, post, par, kind, name)
Heißt im folgenden **accel**.
- Separate Relationen für zusätzliche Information,
z. B. **(pre, text)**, **(pre, attr)**

Umsetzung in SQL (2)

Liegt v im Fenster
und erfüllt es
die anderen Bedingungen?

- Query Window Test:
INSIDE(<[pre_l, pre_h], [post_l, post_h], p, k, n>, v) \equiv

pre_l < v.pre **AND** pre_h > v.pre
AND post_l < v.post **AND** post_h > v.post
AND v.par = p **AND** v.kind = k **AND** v.name = n

- INSIDE hat zwei Parameter, den DESCR Vektor und v.
- Wenn Bedingung nur aus Teil der INSIDE-Klausel besteht, dann entsprechende Bedingungen in WHERE-Klausel einfach weglassen.

Umsetzung in SQL (3)

- Übersetzung des XPath-Ausdrucks

$[/]s_1/s_2/ \dots /s_n$:

```
SELECT DISTINCT  $v_n$ .*
```

```
FROM context  $c$ , accel  $v_1$ , ..., accel  $v_n$ 
```

```
WHERE INSIDE(window( $s_1$ ,  $c$ ),  $v_1$ ) AND ...
```

```
AND INSIDE(window( $s_n$ ,  $v_{n-1}$ ),  $v_n$ )
```

```
ORDER BY  $v_n$ .pre ASC
```

window(s_i , v) –
erzeugt Bedingungsvektor
für Kind-Elemente von v namens s_i

Umsetzung in SQL (3)

- Übersetzung des XPath-Ausdrucks $[/]s_1/s_2/ \dots /s_n$:

```
SELECT DISTINCT vn.*
FROM context c, accel v1, accel v2, ..., accel vn
WHERE INSIDE(window(s1, c), v1)
      AND INSIE (window(s2, v1), v2)
      AND INSIDE(window(sn, vn-1), vn)
ORDER BY vn.pre ASC
```

- Markierte Teile repräsentieren einen Location Schritt, z.B. s_1

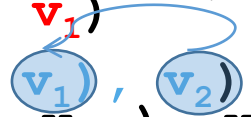
Umsetzung in SQL (3)

- Übersetzung des XPath-Ausdrucks $[/]s_1/s_2/ \dots /s_n$:

```

SELECT DISTINCT vn.*
FROM context c, accel v1, accel v2, ..., accel vn
WHERE INSIDE(window(s1, c), v1)
      AND INSIE (window(s2, v1), v2)
      AND INSIDE(window(sn, vn-1), vn)
ORDER BY vn.pre ASC

```



- Verbindung der Location steps, wie s_1/s_2 , über **joins**
 - s_1 Menge an Input-Knoten (Zwischenresultat)
 - s_2 Knoten, die von s_1 s_2 erfüllen (neues Zwischenresultat)
 - Berechnet pro Location step einen Join
 - Intuition: Man baut sich das Dokument ausgehend von s_1 über die location steps zusammen und gibt am Ende s_n aus

Umsetzung in SQL (3)

- Übersetzung des XPath-Ausdrucks
[/] $s_1/s_2/ \dots /s_n$:
SELECT DISTINCT v_n .*
 FROM context c , accel $v_1, \dots, \text{accel } v_n$
 WHERE INSIDE(window(s_1, c), v_1) AND ...
 AND INSIDE(window(s_n, v_{n-1}), v_n)
ORDER BY v_n .pre ASC
- **Context c : Input \rightarrow Menge der Knoten**
- Absolute Pfade: Nur Wurzelknoten
- Relative Pfade: Menge an Knoten

<u>Pre</u>	Post	Parent	Kind	Type
5	8	0	elem	rezept

Umsetzung in SQL (4)

- Übersetzen Sie
 - den Pfadausdruck $s_1[s_2]/s_3$
 - die XPath-Anfrage
`p=descendant-or-self::n/preceding-sibling::text()`

nach SQL, gegeben das in der Vorlesung eingeführte Relationenschema.

Umsetzung in SQL (4)

- Übersetzen Sie
 - den Pfadausdruck $s_1[s_2]/s_3$
 - die XPath-Anfrage
p=descendant-or-self::n/preceding-sibling::text()

nach SQL, gegeben das in der Vorlesung eingeführte Relationenschema.

```
SELECT DISTINCT v3.*
FROM context c, accel v1, accel v2, accel v3
WHERE INSIDE(window(s1, c), v1)
  AND INSIDE(window(s2, v1), v2)
  AND INSIDE(window(s3, v1), v3)
ORDER BY v3.pre ASC
```


Umsetzung in SQL (4)

- Übersetzen Sie
 - den Pfadausdruck $s_1[s_2]/s_3$
 - die XPath-Anfrage
`p=descendant-or-self::n/preceding-sibling::text()`

nach SQL, gegeben das in der Vorlesung eingeführte Relationenschema.

```
SELECT DISTINCT v3.*
FROM context c, accel v1, accel v2, accel v3
WHERE INSIDE(window(s1, c), v1)
  AND INSIDE(window(s2, v1), v2)
  AND INSIDE(window(s3, v1), v3)
ORDER BY v3.pre ASC
```

Umsetzung in SQL (4)

- Übersetzen Sie
 - den Pfadausdruck $s_1[s_2]/s_3$
 - die XPath-Anfrage
p=descendant-or-self::n/preceding-sibling::text()

nach SQL, gegeben das in der Vorlesung eingeführte Relationenschema.

```
SELECT DISTINCT v2.*
FROM context c, accel v1, accel v2
WHERE c.pre <= v1.pre AND v1.post <= c.post
  AND v1.name = n
  AND v2.pre < v1.pre AND v2.post < v1.post
  AND v2.par = v1.par
  AND v2.kind = text
ORDER BY v2.pre ASC
```

Umsetzung in SQL (4)

- Übersetzen Sie
 - den Pfadausdruck $s_1[s_2]/s_3$
 - die XPath-Anfrage
p=descendant-or-self::n/preceding-sibling::text()

nach SQL, gegeben das in der Vorlesung eingeführte Relationenschema.

```
SELECT DISTINCT v2.*
FROM context c, accel v1, accel v2
WHERE c.pre <= v1.pre AND v1.post <= c.post --descendant
AND v1.name = n -- Nur Knoten vom Typ 'n'
AND v2.pre < v1.pre AND v2.post < v1.post
AND v2.par = v1.par
AND v2.kind = text
ORDER BY v2.pre ASC
```

Umsetzung in SQL (4)

- Übersetzen Sie
 - den Pfadausdruck $s_1[s_2]/s_3$
 - die XPath-Anfrage
`p=descendant-or-self::n/preceding-sibling::text()`

nach SQL, gegeben das in der Vorlesung eingeführte Relationenschema.

```
SELECT DISTINCT v2.*
FROM context c, accel v1, accel v2
WHERE c.pre <= v1.pre AND v1.post <= c.post --descendant
      AND v1.name = n -- Nur Knoten vom Typ 'n'
      AND v2.pre < v1.pre AND v2.post < v1.post --preceding
      AND v2.par = v1.par --nur Knoten auf der selben Ebene
      AND v2.kind = text --nur Text Knoten
ORDER BY v2.pre ASC
```

Schlussbemerkungen

- Indexstrukturen für XPath relational implementiert. ‚relational index structure‘.
- Abbildung XML → Relationen nicht offensichtlich, aber recht elegant.

Prüfungsfragen, beispielhaft (1)

- Welche Möglichkeiten der Speicherung von semistrukturierten Datenbeständen in relationalen Datenbanken kennen Sie?
- Wie funktioniert Ansatz X? Welche Anfragen werden gut unterstützt? Was sind die Grenzen von Ansatz X?

Prüfungsfragen, beispielhaft (2)

- Was ist das EDGE-Modell?
Warum stellt es uns nicht zufrieden?
- Wieso ist es vorteilhaft, die Struktur
eines Dokumentbestands für die Speicherung/
für die Evaluierung von Anfragen zu kennen?

Prüfungsfragen, beispielhaft (3)

- Warum eignet sich der XPath Accelerator für Realisierung mit RDBMS Technologie?
- <Umsetzung des XPath Accelerators für einen Location Step nach SQL wiedergeben können.>
- <Eine Optimierung des XPath Accelerators erklären können.>

Literatur (2)

- Jayavel Shanmugasundaram et al.: **Relational Databases for Querying XML Documents: Limitations and Opportunities.** VLDB 1999
– einzelne Aspekte wurden hier wiedergegeben –
- D. Grossman and O. Frieder,
**Information Retrieval:
Algorithms and Heuristics,**
Kluwer Academic Publishers, ISBN 0-7923-8271-4, 1998.
Zweite Ausgabe 2004 im Springer Verlag.
- T. Grust, J. Teubner, M. van Keulen.
Accelerating XPath Evaluation in Any RDBMS,
in *ACM Transactions on Database Systems (TODS)*, 29(1),
März 2004.