



**Hasso  
Plattner  
Institut**

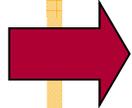
IT Systems Engineering | Universität Potsdam

Datenbanksysteme I  
XML & Datenbanken

12.7.2010

Felix Naumann

2



- Motivation & Syntax
- XML Prozessoren
- Schemata
- Anfragesprachen
- Speicherung von XML



# Motivation

3

- XML - EXtensible Markup Language
- *mark up* – ursprünglich aus dem Verlagswesen, Anweisungen an den Setzer
- Daten und Informationen über die Daten (Struktur/Metadaten) im gleichen Dokument
- Durch das World Wide Web Consortium (W3C) entwickelt
- Gut lesbar (human readable)
  - Ziel nicht erreicht
- Häufig eingesetztes Austauschformat

# Syntax von Elementen

4

- „Grundbausteine“ eines XML-Dokumentes
- Ein Element besteht aus:
  - Start-Tag
  - Ende-Tag
  - Elementinhalt



```
<vortragender> Ronald Bourret </vortragender>
```

Start-Tag      Elementinhalt      Ende-Tag

# Syntax von Elementen

5

- Leere Elemente:

```
<koordinaten/>  
<koordinaten></koordinaten>
```

- Schachtelung:

```
<vortragender>  
  <name>Bourret</name>  
  <vorname>Ronald</vorname>  
</vortragender>
```

Start-Tag

Elementinhalt

Ende-Tag

# Klassifikation von XML-Dokumenten

6

- Datenzentrierte Dokumente
  - strukturiert, regulär
  - Beispiele: Produktkataloge, Bestellungen, Rechnungen
- Dokumentzentrierte Dokumente
  - unstrukturiert, irregulär
  - Beispiele: wissenschaftliche Artikel, Bücher, E-Mails, Webseiten
- Semistrukturierte Dokumente
  - datenzentrierte und dokumentzentrierte Anteile
  - Beispiele: Veröffentlichungen, Amazon

```
<order>
  <customer>Meyer</customer>
  <position>
    <isbn>1-234-56789-0</isbn>
    <number>2</number>
    <price currency=„Euro“>30.00</price>
  </position>
</order>
```

```
<content>
XML builds on the principles of two existing
languages, <emph>HTML</emph> and
<emph>SGML</emph> to create a simple
mechanism ..
The generalized markup concept ..
</content>
```

```
<book>
  <author>Neil Bradley</author>
  <title>XML companion</title>
  <isbn>1-234-56789-0</isbn>
  <content>
    XML builds on the principles of two existing
    languages, <emph>HTML</emph> and ..
  </content>
</book>
```

# Datenzentriertes XML-Dokument

7

```
<?xml version="1.0" encoding="UTF-8"?>
<rechnung kundenummer="k333063143">
  <monatspreis>0,00</monatspreis>
  <einzelverbindungsachweis>
    <verbindung>
      <datum>26.2.</datum>
      <zeit>19:47</zeit>
      <nummer>200xxxx</nummer>
      <einzelpreis waehrung="Euro">0,66</einzelpreis>
    </verbindung>
    <verbindung>
      <datum>27.2.</datum>
      <zeit>19:06</zeit>
      <nummer>200xxxx</nummer>
      <einzelpreis waehrung="Euro">0.46</einzelpreis>
    </verbindung>
    <verbindungskosten_gesamt waehrung="Euro">2.19</verbindungskosten_gesamt>
  </einzelverbindungsachweis>
</rechnung>
```

# XML-Dokument, Eigenschaften

8

- Selbstbeschreibend
  - XML-Dokumente enthalten Daten und Struktur über die Daten in einem Dokument.
- Irregulär (semistrukturiert)
  - Alle Dokumente sind unterschiedlich strukturiert
- Ungetypt
  - Informationen im XML-Dokument haben keinen oder einen wechselnden Datentyp
- Dokumentzentriert
  - XML-Dokumente enthalten große Anteile von Volltext

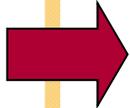
# XML vs. Datenbanken

9

- Datenbanken für XML Daten
  - Welche Speicherungstechniken für XML-Dokumente?
  - Welche Indizierung für gespeicherte Dokumente?
  - Welche Anfragesprache und Updatesprache?
  - Abbildung von XML zum relationalen Modell
- XML Daten aus Datenbanken
  - Neue Anforderungen an DBMS
  - XML Erweiterungen in kommerziellen DBMS
  - Abbildung von Relationen auf das XML Datenmodell

10

- Motivation & Syntax
- XML Prozessoren
- Schemata
- Anfragesprachen
- Speicherung von XML



# XML-Prozessoren

11

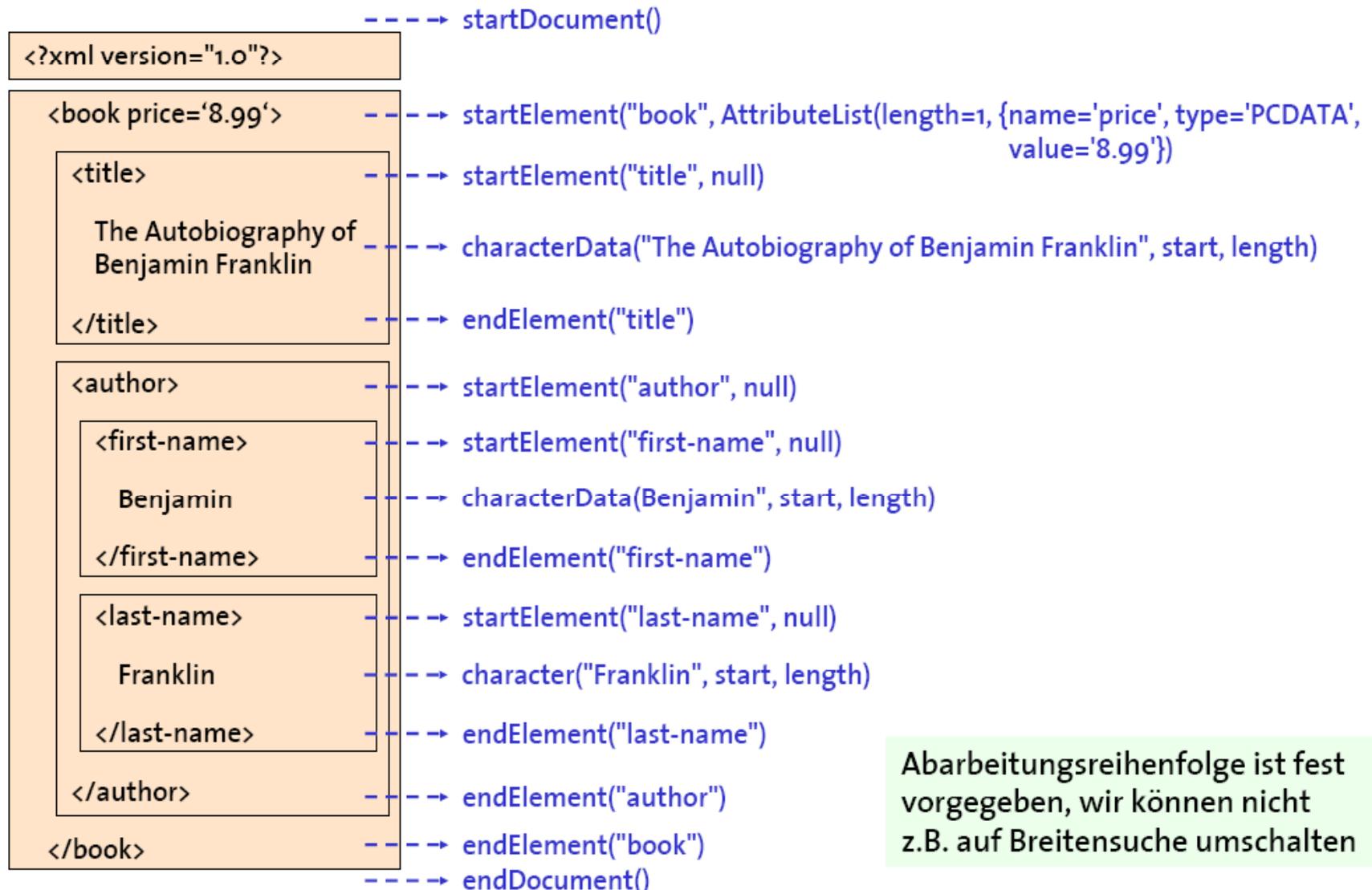
- Inhalt eines XML-Dokumentes wird für eine Anwendung verfügbar.
- Standardisierte Schnittstellen für zahlreiche Programmiersprachen
  - Java, Python, C, C++, ...
- Zwei prominente Vertreter:
  - Ereignis-orientiert: SAX
  - Baum-orientiert: DOM

# SAX - Simple API for XML

12

- Ereignisorientierte Verarbeitung
- Vorgehensweise:
  - XML-Engine liest sequentiell den Eingabestrom (das Dokument) und
  - ruft Callback-Methoden bei Eintreten von Ereignissen auf,
    - ◇ z.B. wenn ein Begin oder End-Tag abgearbeitet wird.
  - Anwendung kann auf diese Ereignisse reagieren oder sie ignorieren.
  - Anwendungsprogrammierer muss "Event-Handler" für die Callback-Methoden implementieren, an deren Ereignissen er interessiert ist.
  - SAX ist zustandslos.

# SAX – Beispiel



Abarbeitungsreihenfolge ist fest vorgegeben, wir können nicht z.B. auf Breitensuche umschalten

# DOM – Document Object Model

14

- DOM
  - Beschreibt Schnittstellen (APIs) zum Zugriff auf XML-Dokumente und zur Veränderung von Struktur und Inhalten
  - Definiert nicht die zugrunde liegende Implementierung und Speicherung der XML-Dokumente
- Objektorientierte Sicht auf XML-Dokumente
  - XML-Dokumente intern als Bäume repräsentiert
  - Unterschiedliche Knotentypen: Element, Attribut, etc.
  - Methoden zum Traversieren und Manipulieren der Baumstruktur
- Es gibt zahlreiche DOM-Implementierungen, z.B.
  - Java (+ XML-Parser)
  - JavaScript und Web-Browser
  - C++-Bibliotheken

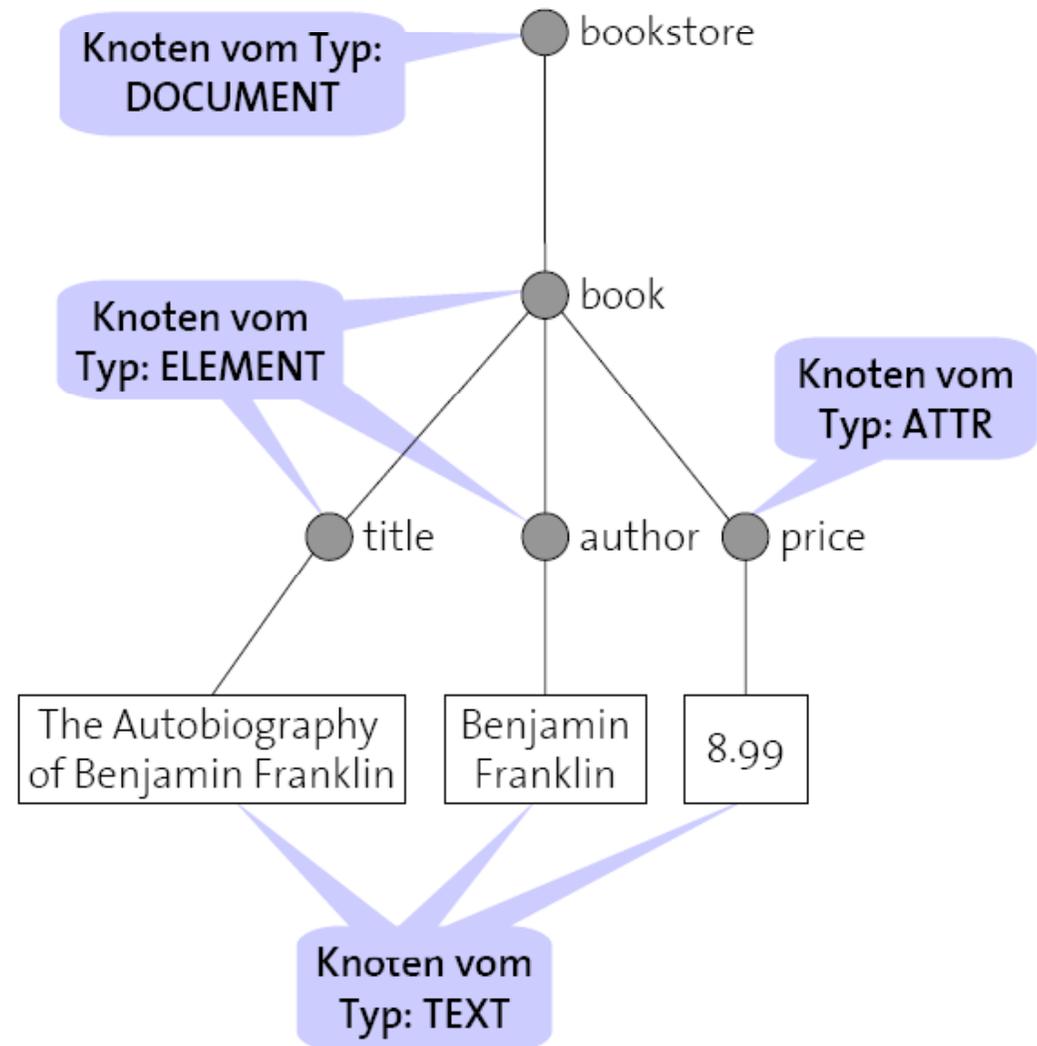
Quelle: Can Türker

# DOM – als Baum

15

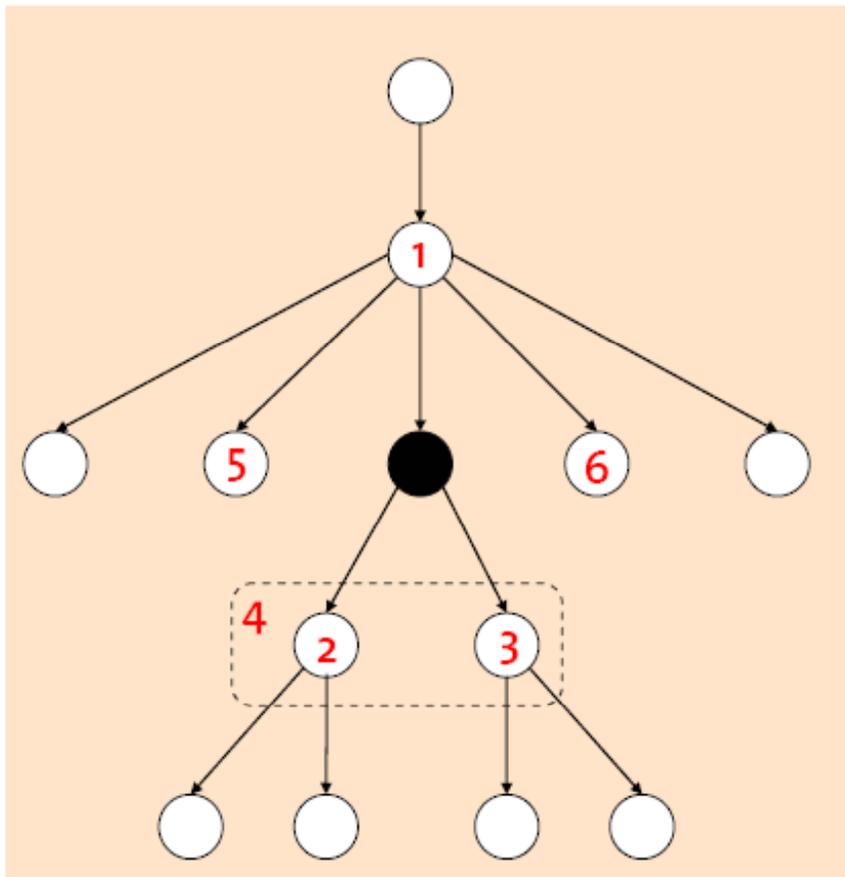
```

<?xml version="1.0"?>
<bookstore>
  <book price='8.99'>
    <title>
      The Autobiography of
      Benjamin Franklin
    </title>
    <author>
      Benjamin Franklin
    </author>
  </book>
</bookstore>
  
```



# DOM – Navigation durch das Dokument

16

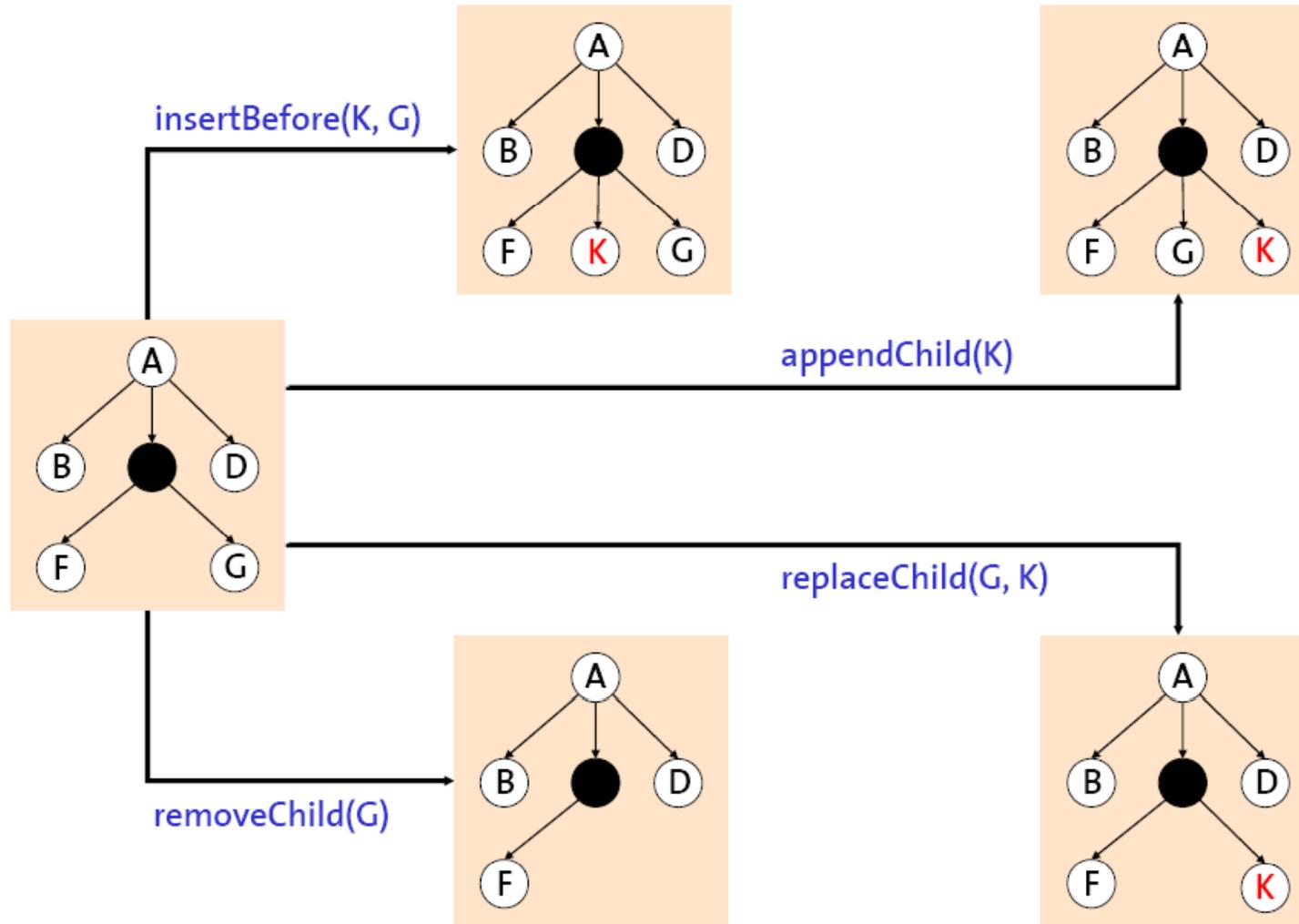


Ausgehend vom Knoten ●  
liefern folgende Methoden der Klasse Node  
die Knoten bzw. Knotenlisten als Ergebnis:

- 1 - `getParentNode()`
- 2 - `getFirstChild()`
- 3 - `getLastChild()`
- 4 - `getChildren()`
- 5 - `getPreviousSibling()`
- 6 - `getNextSibling()`

# DOM – Manipulation der Struktur

17



# Vergleich von SAX und DOM

18

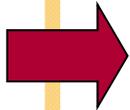
## ■ SAX

- Einfacher Zugriff
- Einfach strukturierte oder gleichartig strukturierte Dokumente
- Geeignet für sehr große XML-Dokumente
- Geeignet, wenn Zugriff nur auf geringe Anteile eines Dokumentes erfolgt

## ■ DOM

- Navigation durch Dokumentstruktur
- Dadurch kontextabhängige Zugriffe
- Manipulation der Struktur
- Für sehr große XML-Dokumente problematisch
  - ◇ Faktor ca. 10

- Motivation & Syntax
- XML Prozessoren
- Schemata
- Anfragesprachen
- Speicherung von XML



# DTDs

20

```
<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE hotel SYSTEM "hotel_dt.dtd">
<hotel id="id001"
  url="http://www.hotel-huebner.de">
  <name>Strand Hotel Huebner</name>
  <adresse>
    <plz>18119</plz>
    <ort>Rostock-Warnemuende</ort>
    ...
  </adresse>
  <hausbeschreibung> Direkt an der
  Promenade von Warnemuende befindet sich das
  Strand-Hotel Huebner mit Blick auf Leuchtturm,
  Hafeneinfahrt und Strand.
  </hausbeschreibung>
  <preise waehrung="Euro">
    <einzelzimmer>ab 78,-</></>
  ...
</hotel>
```

```
<!-- Hotel DTD-->
<!ELEMENT hotel (name, kategorie?,
  adresse, hausbeschreibung, preise*)>
<!ATTLIST hotel id ID #REQUIRED
  url CDATA #REQUIRED>
<!ELEMENT name (#PCDATA)>
<!ELEMENT kategorie (#PCDATA)>
<!ELEMENT adresse (plz, ort, strasse,
  hausnummer, telefon, fax?, e-mail?)>
<!ELEMENT plz (#PCDATA)>
...
<!ELEMENT hausbeschreibung (#PCDATA)>
<!ELEMENT preise (#PCDATA | einzelzimmer
  | doppelzimmer | apartment)*>
<!ATTLIST preise waehrung CDATA
  #REQUIRED>
<!ELEMENT einzelzimmer (#PCDATA)>
...
```

# XML Schema

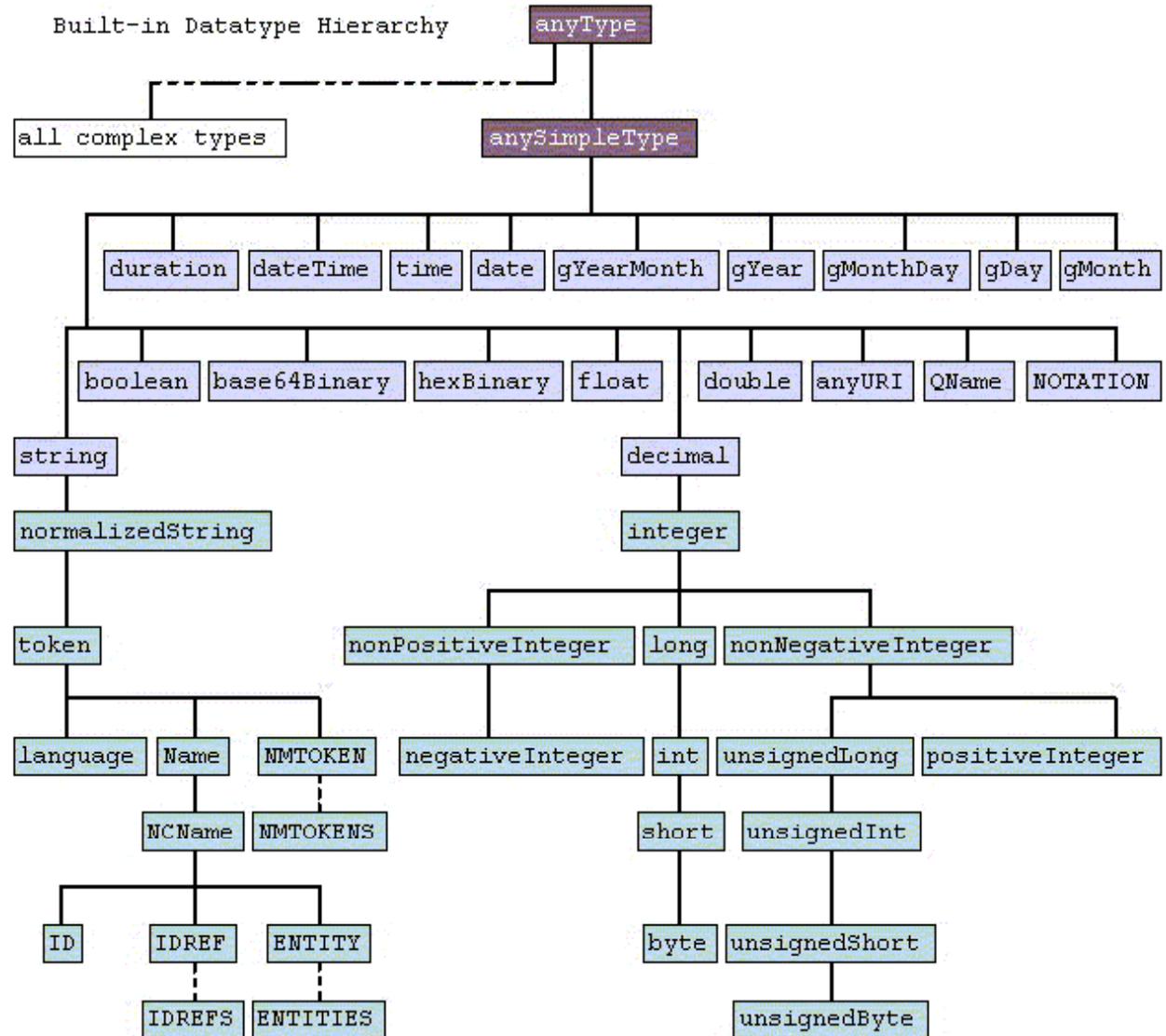
21

Wesentlich umfangreichere Darstellungsmöglichkeiten als DTDs

- Vielfältige vordefinierte Datentypen
- Definition eigener Datentypen
- Umfangreiche Darstellungsmöglichkeiten
- Integritätsbedingungen
  - Unique, Key, Foreign key
- Dargestellt in XML-Syntax
  - Leichter parsebar

# Typhierarchie von XML Schema

22



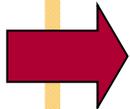
- ur types
- built-in primitive types
- built-in derived types
- complex types
- derived by restriction
- derived by list
- derived by extension or restriction

## XML Schema – Beispiel

23

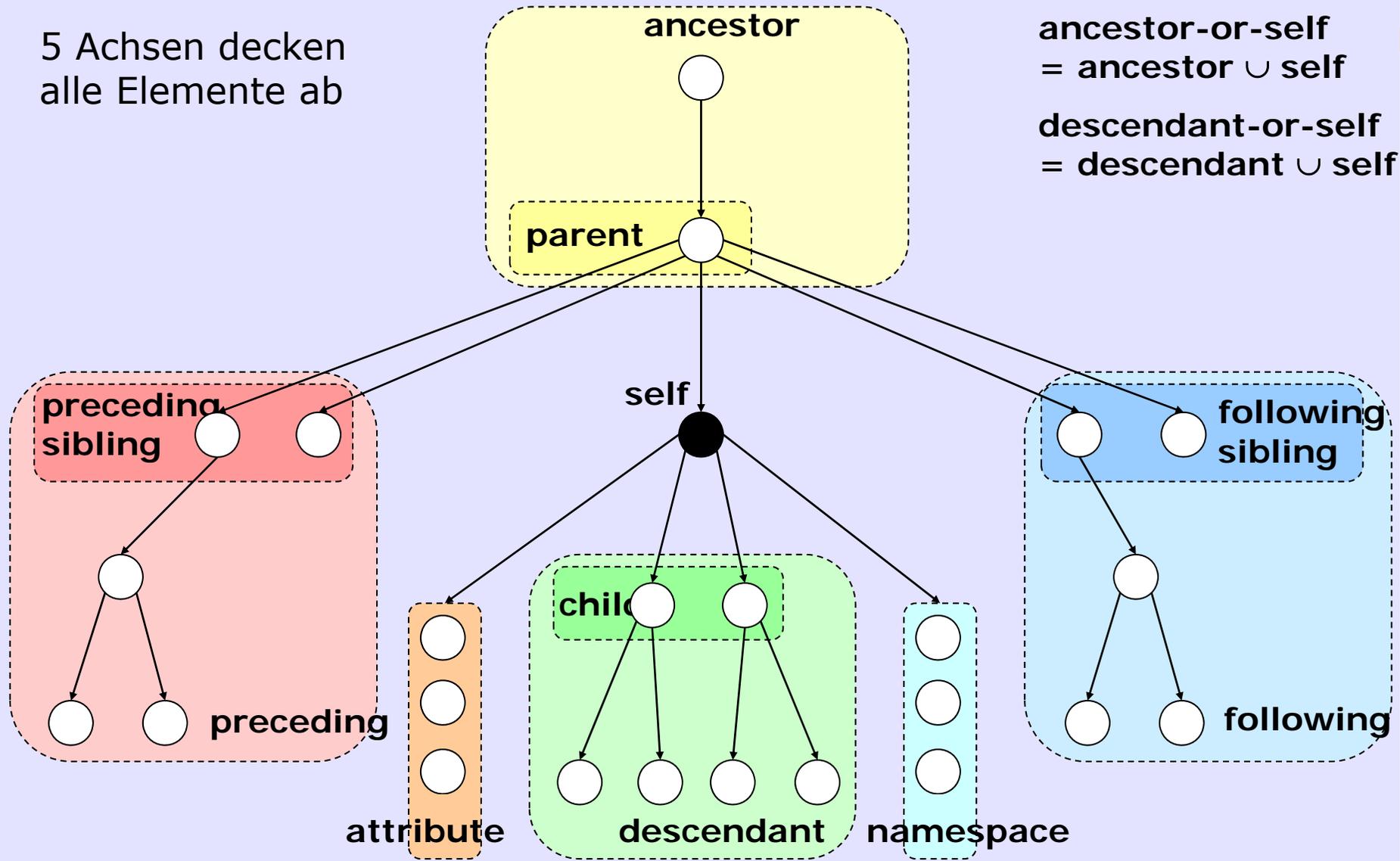
```
<xs:complexType name="adresseType">
  <xs:sequence>
    <xs:element ref="plz"/>
    <xs:element ref="ort"/>
    <xs:choice>
      <xs:sequence>
        <xs:element ref="strasse"/>
        <xs:element ref="nummer"/>
      </xs:sequence>
      <xs:element ref="postfach"/>
    </xs:choice>
    <xs:element ref="telefon"/>
    <xs:element ref="fax" minOccurs="0"/>
  </xs:sequence>
</xs:complexType>
```

- Motivation & Syntax
- XML Prozessoren
- Schemata
- Anfragesprachen
  - Xpath, Xquery, XSLT
- Speicherung von XML



# XPath – Navigationsachsen

5 Achsen decken alle Elemente ab



**ancestor-or-self**  
= ancestor  $\cup$  self

**descendant-or-self**  
= descendant  $\cup$  self

## Beispiele für XPath-Anfragen

26

- `/bookstore/book/@genre`
  - Gibt das Attribut 'Genre' aller Bücher aus
- `/bookstore/book[author/name='Plato']`
  - Alle Bücher, die vom Autor 'Plato' stammen
- `//author[first-name='Herman']/last-name`
  - Nachnamen aller Autoren, deren Vorname 'Herman' ist
- `/bookstore/book[author/first-name='Benjamin']/price`
  - Preis für alle Bücher, die mind. einen Autor mit dem Vornamen 'Benjamin' haben
- `//book[contains(title, 'XML')]/title`
  - Alle Bücher, die den Begriff 'XML' im Titel enthalten

- Basiert auf XPath
- Ähnlichkeit zu SQL
- Basiskonstrukt: FLWR-Ausdruck,
  - **for/let**: geordnete Liste von Elementen
  - **where**: eingeschränkte Liste von Elementen
  - **return**: Ergebniskonstruktion, Instanzen des XML Query data model
- Ausdrücke werden aus anderen Ausdrücken zusammengesetzt.
- Datenmodell ist geordneter Wald
  - Flach
  - Geordnet
  - Nicht duplikatfrei

# Beispiel

28

```
<hotel name="Hotel Neptun">
  <zimmertyp typ="EZ" preis="180" waehrung="DM" />
  <foto href="neptun01.jpeg" />
</hotel>
<hotel name="Hotel Huebner">
  <zimmertyp typ="EZ" preis="150" waehrung="DM" />
  <zimmertyp typ="DZ" preis="180" waehrung="DM" />
</hotel>
<hotel name="Pension Draeger">
  <foto href="bild-pd01.jpeg" />
  <foto href="bild-pd02.jpeg" />
</hotel>
```

```
for $hotel in //hotel
return $hotel/foto
```

```
<foto href="neptun01.jpeg" />
<foto href="bild-pd01.jpeg" />
<foto href="bild-pd02.jpeg" />
```

Auswertung von return  
pro hotel, aber nicht immer  
mit Ergebnis.

# Beispiel

{ } kennzeichnet einen auszuwertenden Ausdruck

29

```
<billighotels> {  
  for $h in //hotel  
  for $z in $h/zimmertyp  
  where $z/@preis <= 100  
  return <hotel>  
    <name>{ $h/@name }</name>  
    <preis>{ $z/@preis }</preis>  
  </hotel> }  
</billighotels>
```

```
<billighotels>  
  <hotel>  
    <name>...</name>  
    <preis>...</preis>  
  </hotel>  
  ...  
</billighotels>
```

Ein hotel-Element pro zimmertyp

# Beispiele

30

```

<result>
  {
    for $b in fn:doc("bib.xml")/bib/book
    where $b/publisher = "Addison-Wesley" and $b/@year > "1991"
    return <book year="{ $b/@year }">
      { $b/title }
      </book>
  }
</result>

```

Welche Bücher sind von Addison-Wesley nach 1991 publiziert worden?

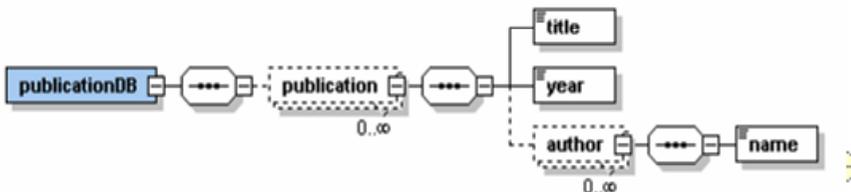
```

<result>
  <book year="1994">
    <title>TCP/IP Illustrated</title>
  </book>
  <book year="1992">
    <title>Advanced Programming in the Unix environment</title>
  </book>
</result>

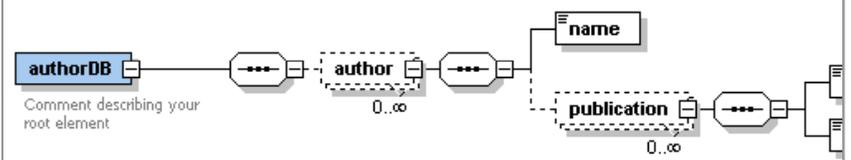
```

# Beispiel

21



Pivotisierung



```
LET $doc0 := document("input XML file goes here")  
RETURN
```

```
<authorDB>
```

```
{  
  distinct-values (  
    FOR
```

```
    $x0 IN $doc0/publicationDB/publication,  
    $x1 IN $x0/author
```

```
  RETURN
```

```
    <author>
```

```
      <name> { $x1/name/text() } </name>
```

```
      {  
        distinct-values (  
          FOR
```

```
            $x0L1 IN $doc0/publicationDB/publication,  
            $x1L1 IN $x0L1/author
```

```
          WHERE
```

```
            $x1/name/text() = $x1L1/name/text()
```

```
          RETURN
```

```
            <publication>
```

```
              <title> { $x0L1/title/text() } </title>
```

```
              <year> { $x0L1/year/text() } </year>
```

```
            </publication> )
```

```
      }
```

```
    </author> )
```

```
  }
```

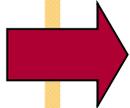
```
</authorDB>
```

# Vergleich XQuery / SQL

32

XQuery	SQL
for \$k in /bookstore/book return \$k	SELECT * FROM bookstore
for \$k in //book return \$k	SELECT * FROM bookstore
for \$k in //book/title return \$k	SELECT title FROM bookstore
for \$k in //book return \$k/title	SELECT title FROM bookstore
for \$k in //book/author return \$k/last-name	SELECT last-name FROM bookstore
for \$k in /bookstore/book where \$k/title='XML und Datenbanken' order by \$k/author/last-name return \$k/author	SELECT author FROM bookstore WHERE title='XML und Datenbanken' ORDER BY author.last-name
for \$k in /bookstore/book where count(\$k/author) > 2 return \$k/title	SELECT title FROM bookstore GROUP BY title HAVING COUNT(author) > 2

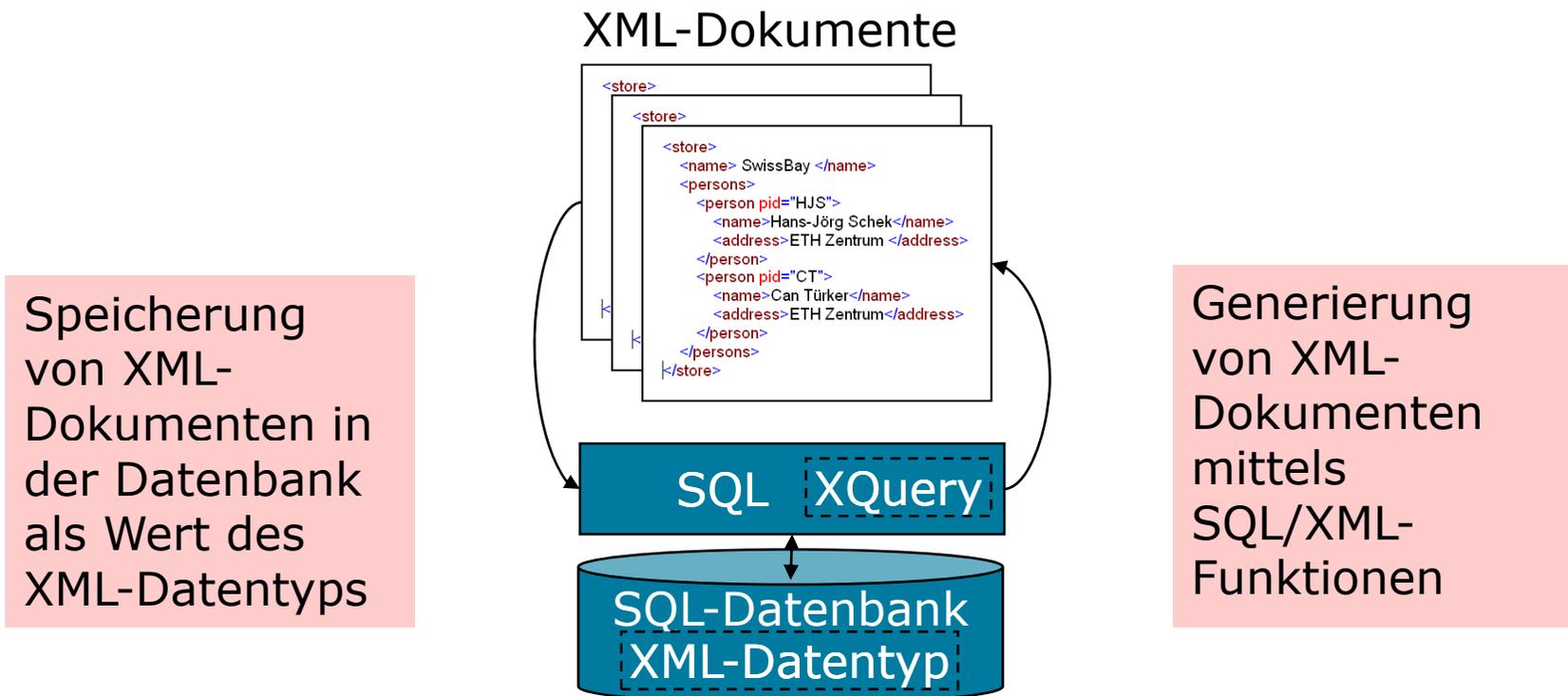
- Motivation & Syntax
- XML Prozessoren
- Schemata
- Anfragesprachen
- Speicherung von XML
  - SQL/XML
  - Modell-basierte Verfahren
  - Struktur-basierte Verfahren
  - Indizierung



# „SQL / XML“ – Grundidee

34

- SQL/XML stellt neuen Datentyp XML mit darauf operierenden Funktionen bereit.
- Definiert Abbildungen zwischen SQL und XML



# Basisdatentyp XML

35

- Ermöglicht Speicherung von XML-Werten in Tabellenspalten.
- XML-Werte:

- NULL
- XML-Dokument
  - ◇ mit oder ohne Prolog
- XML-Element
- Wald von XML-Elementen
  - ◇ Keine eindeutige Wurzel

ID	Autorennamen
12	<Autoren>Jim Beam</Autoren>
23	<Autoren>Johnny Walker</Autoren>
87	<Autoren>Jack, Jim Beam</Autoren>

- **CREATE TABLE Angestellte**

```
(
  ID          INTEGER,
  Gehalt     DECIMAL(12,2),
  Bewerbung  XML
);
```

# Basisdatentyp XML

36

- Element mit Attribut:
  - `<vortragender tutorial='T1'> Ronald Bourret  
</vortragender>`
- Element mit Subelement:
  - `<vortragender> <name>Bourret</name>  
<vorname>Ronald</vorname> </vortragender>`
- Mixed content:
  - `<vortragender>Prof. <vorname>Andreas</vorname>  
<name>Heuer</name>, Rostock</vortragender>`
- Wald:
  - `<vorname>Ronald</vorname> <name>Bourret</name>`
- XML-Dokument
  - `<?XML version="1.0" encoding="UTF-8",  
standalone="yes">  
<persons>  
<name>Bourret </name><vorname>Ronald</vorname>  
</persons>`

# XML-Funktionen

37

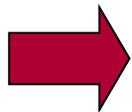
- XMLGEN
  - generiert ein XML-Dokument mittels einer XQuery Anfrage
- XMLELEMENT
  - erzeugt ein XML-Element aus einer Werteliste
  - XMLATTRIBUTES erzeugt dazu XML-Attribute
- XMLFOREST
  - erzeugt aus beliebigen Werten einen Wald von XML-Elementen
- XMLCONCAT
  - konkateniert mehrere XML-Elemente zu einem Wald
- XMLAGG
  - aggregiert die XML-Elemente einer Gruppe

# Klassifikation der Ansätze für die physische Ebene

38

- Textbasierte Verfahren
  - Speicherung der XML-Dokumente als Zeichenkette
- Modellbasierte Verfahren
  - Ausnutzen des Datenmodells von XML zur Speicherung
  - Generische Speicherung der Graphstruktur von XML
  - Speicherung der DOM-Informationen
- Strukturbasierte Verfahren
  - Abbildung auf relationale Datenbanken
  - Ableiten des DB-Schemas aus XML-Struktur
  - Einsatz von benutzerdefinierten Abbildungsverfahren

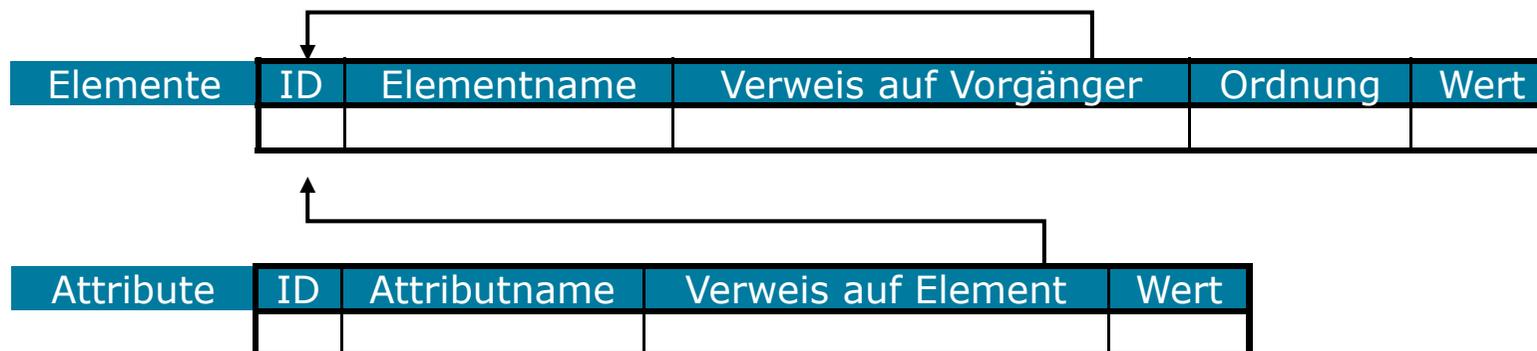
- Motivation & Syntax
- XML Prozessoren
- Schemata
- Anfragesprachen
- Speicherung von XML
  - SQL/XML
  - Modell-basierte Verfahren
  - Struktur-basierte Verfahren
  - Indizierung



# Modellbasierte Speicherung

40

- Idee: generische Speicherung der Graphstrukturen von XML-Dokumenten
  - XML-Elemente, XML-Attribute, ... sind die Knoten des Graphen
  - Schachtelung der Elemente sind die Kanten
  - Knoten erhalten (intern) eine ID durch Traversierung des Graphen
- Verwendung von Relationen zur Speicherung von Elementen und Attributen



- Vollständige Wiederherstellung der Struktur ist möglich.

# Beispiel

41

```
<hotel id=„H0001“  
  url=http://www.hotel-huebner.de  
  erstellt-am=„09/15/2002“  
  autor=„Hans Huebner“>  
<hotelname>Hotel Huebner</hotelname>  
<kategorie>4</kategorie>  
<adresse>  
  <plz>18119</plz>  
  <ort>Warnemuende</ort>  
  <strasse>Seestrasse</strasse>  
  <nummer>12</nummer>  
</adresse>  
<telefon>0381 / 5434-0</telefon>  
<fax>0381 / 5434-444</fax>  
</hotel>
```

# Beispiel

42

## Elemente

DocID	Elementname	ID	Vorgaenger	Ordnung	Wert
H00001	Hotel	101	⊥	1	⊥
H00001	Hotelname	102	101	1	Hotel Huebner
H00001	Kategorie	103	101	2	4
H00001	Adresse	104	101	3	⊥
H00001	PLZ	105	104	1	18119
H00001	Ort	106	104	2	Warnemuende
H00001	Strasse	107	104	3	Seestrasse
H00001	Nummer	108	104	4	12
H00001	Telefon	109	101	4	0381 / 5434-0
H00001	Fax	110	101	5	0381 / 5434-444

# Anfragen

43

- Angepasstes SQL, durch Datenbankschema bestimmt:

- Beispiel:

- Hotels in Warnemünde

```

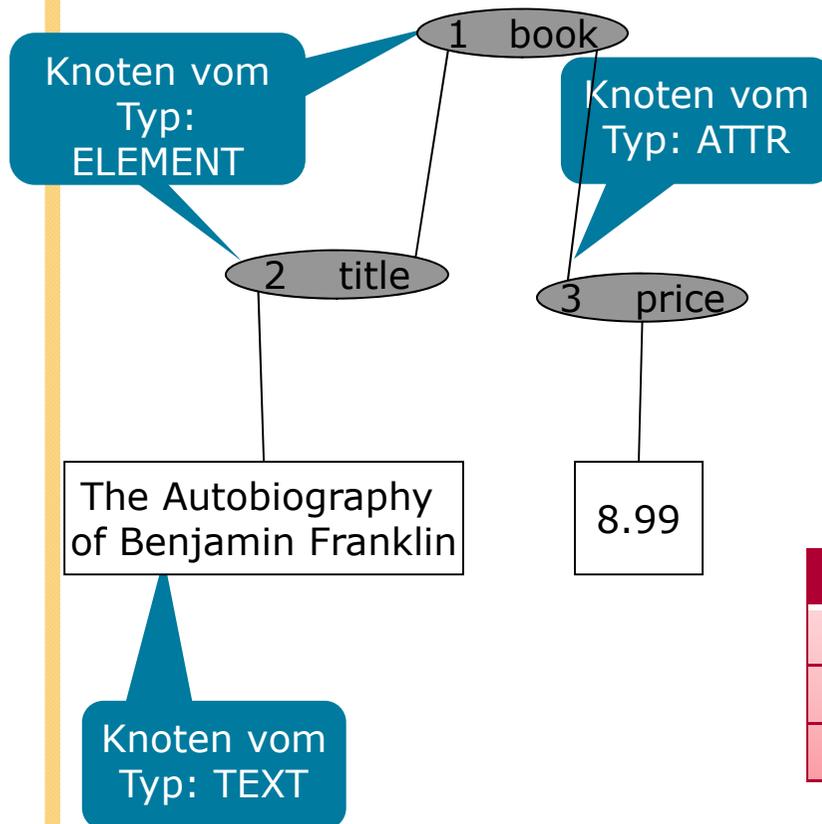
□ SELECT a.wert
FROM Elements a,
Elements b
WHERE a.element =
'hotelname'
AND b.element = 'ort',
AND b.wert =
'Warnemünde'
AND a.DocID = b.DocID

```

DocID	Elementname	ID	Vorgänger	Ordnung	Wert
H00001	Hotel	101	⊥	1	⊥
H00001	Hotelname	102	101	1	Hotel Huebner
H00001	Kategorie	103	101	2	4
H00001	Adresse	104	101	3	⊥
H00001	PLZ	105	104	1	18119
H00001	Ort	106	104	2	Warnemünde
H00001	Strasse	107	104	3	Seestrass e
H00001	Nummer	108	104	4	12
H00001	Telefon	109	101	4	0381 / 5434-0
H00001	Fax	110	101	5	0381 / 5434-444

# Modellbasierte Speicherung: Speicherung des DOM

44



NodeID	NodeType	DocID	ParentNode
1	ELEMENT	1	NULL
2	ELEMENT	1	1
3	ATTRIBUTE	1	1
4	TEXT	1	2

NodeID	TagName
1	book
2	title

NodeID	Content
4	"The Auto..."

NodeID	PreviousSibling	NextSibling	FirstChild
1	NULL	NULL	2
2	NULL	3	4
3	2	NULL	NULL

NodeID	ElementID	AttributName	AttributValue
3	1	price	8.99

# Modellbasierte Speicherung: Speicherung des DOM

45

Primärtabelle: Node

NodeID	NodeType	DocID	ParentNode
1	ELEMENT	1	NULL
2	ELEMENT	1	1
3	ATTRIBUTE	1	1
4	TEXT	1	2

Elemente

NodeID	TagName
1	book
2	title

NodeID	Content
4	"The Auto..."

Elementinhalte

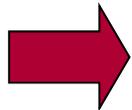
Ordnung

NodeID	PreviousSibling	NextSibling	FirstChild
1	NULL	NULL	2
2	NULL	3	4
3	2	NULL	NULL

Attribute

NodeID	ElementID	AttributName	AttributValue
3	1	price	8.99

- Motivation & Syntax
- XML Prozessoren
- Schemata
- Anfragesprachen
- Speicherung von XML
  - SQL/XML
  - Modell-basierte Verfahren
  - Struktur-basierte Verfahren
  - Indizierung



# Abbildung von XML auf relationale Datenbanken

47

```
<Hotel>
  <HotelID>H0001</HotelID>
  <Name>Hotel Hübner</Name>
  <Adresse>
    <PLZ>18119</PLZ>
    <Ort>Warnemünde</Ort>
    <Strasse>Seestrasse</Strasse>
    <Nr>12</Nr>
  </Adresse>
  <Preise>
    <Einzelzimmer> 198 </Einzelzimmer>
    <Doppelzimmer> 299 </Doppelzimmer>
  </Preise>
</Hotel>
```

Hotel:

HotelID	Hotelname	Adresse	Preise
H0001	Hotel Hübner	A0001	P0001

Adresse:

AdresseID	PLZ	Ort	Strasse	Nr
A0001	18119	Warnemünde	Seestraße	12

Preise:

PreiseID	Einzelzimmer	Doppelzimmer
P0001	198	299

- DTD/XML-Schema ist typischerweise erforderlich.
- Anfragen verwenden SQL-Funktionalität.
- RDBMS-Datentypen werden eingesetzt
- Abbildung von Kollektionstypen durch Aufteilung auf zusätzliche Relationen

# Variante 1 – Alle Alternativen in einer Tabelle

48

<!ELEMENT unterkunft (hotel | pension | campingplatz)\*>

```
<Pension>
  <Name>
    Zum Kater
  </Name>
  <Zimmer>42</Zimmer>
</Pension>
<Hotel>
  <Kategorie>4</Kategorie>
  <Hotelname>
    Hotel Hübner
  </Hotelname>
</Hotel>
<Campingplatz>
  <Sterne>4</Sterne>
  <Name>Meerblick</Name>
</Campingplatz>
```

Hotel_Kategorie	Hotel_Hotelname	Pension_Name	Pension_Zimmer	Camping_Sterne	Camping_Name
NULL	NULL	Zum Kater	42	NULL	NULL
4	Hotel Hübner	NULL	NULL	NULL	NULL
NULL	NULL	NULL	NULL	4	Meerblick

- Probleme:
  - viele NULL-Werte (Speicherplatzverschwendung)
  - Bedeutung einer Zeile nur implizit

# Variante 2 – Aufspaltung auf mehrere Tabellen

49

`<!ELEMENT unterkunft (hotel | pension | campingplatz)*>`

```
<Pension>
  <Name>
    Zum Kater
  </Name>
  <Zimmer>42</Zimmer>
</Pension>
<Hotel>
  <Kategorie>4</Kategorie>
  <Hotelname>
    Hotel Hübner
  </Hotelname>
</Hotel>
<Campingplatz>
  <Sterne>4</Sterne>
  <Name>Meerblick</Name>
</Campingplatz>
```

Kategorie	Hotelname
4	Hotel Hübner

Name	Zimmer
Zum Kater	42

Sterne	Name
4	Meerblick

Bei Anfragen Vereinigung der Tabellen zur Zusammenführung nötig

# Variante 3 – Verwendung einer Spalte vom Typ XML

50

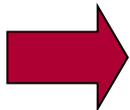
```
<!ELEMENT unterkunft (hotel | pension | campingplatz)*>
```

```
<Pension>
  <Name>
    Zum Kater
  </Name>
  <Zimmer>42</Zimmer>
</Pension>
<Hotel>
  <Kategorie>4</Kategorie>
  <Hotelname>
    Hotel Hübner
  </Hotelname>
</Hotel>
<Campingplatz>
  <Sterne>4</Sterne>
  <Name>Meerblick</Name>
</Campingplatz>
```

XML-Typ bietet native Methoden zum Zugriff mit XML-Anfragen oder DOM Methoden auf die Daten.

```
Unterkunft
<Pension>
  <Name>
    Zum Kater
  </Name>
  <Zimmer>42</Zimmer>
</Pension>
<Hotel>
  <Kategorie>4</Kategorie>
  <Hotelname>
    Hotel Hübner
  </Hotelname>
</Hotel>
<Campingplatz>
  <Sterne>4</Sterne>
  <Name>Meerblick</Name>
</Campingplatz>
```

- Motivation & Syntax
- XML Prozessoren
- Schemata
- Anfragesprachen
- Speicherung von XML
  - SQL/XML
  - Modell-basierte Verfahren
  - Struktur-basierte Verfahren
  - Indizierung



# Pre- und Postorder in XML Bäumen

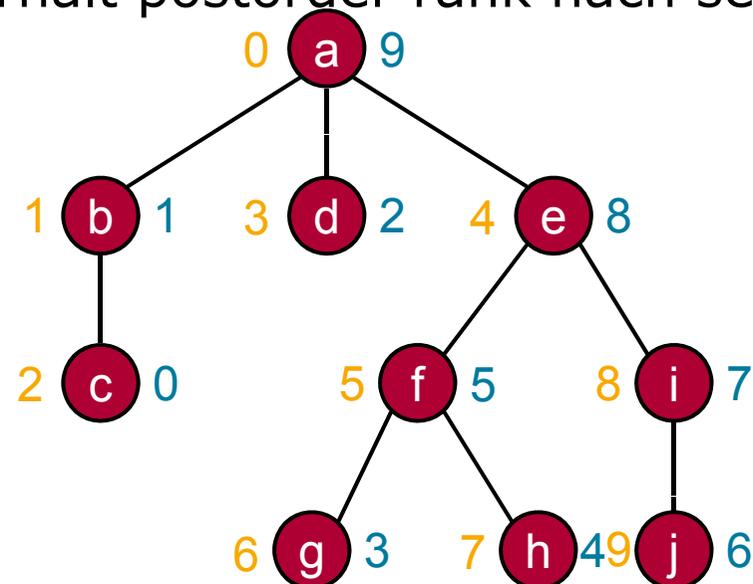
52

## ■ Preorder

- Knoten erhält preorder rank vor seinen Kindern.
- Tiefensuche (*depth-first-search*)

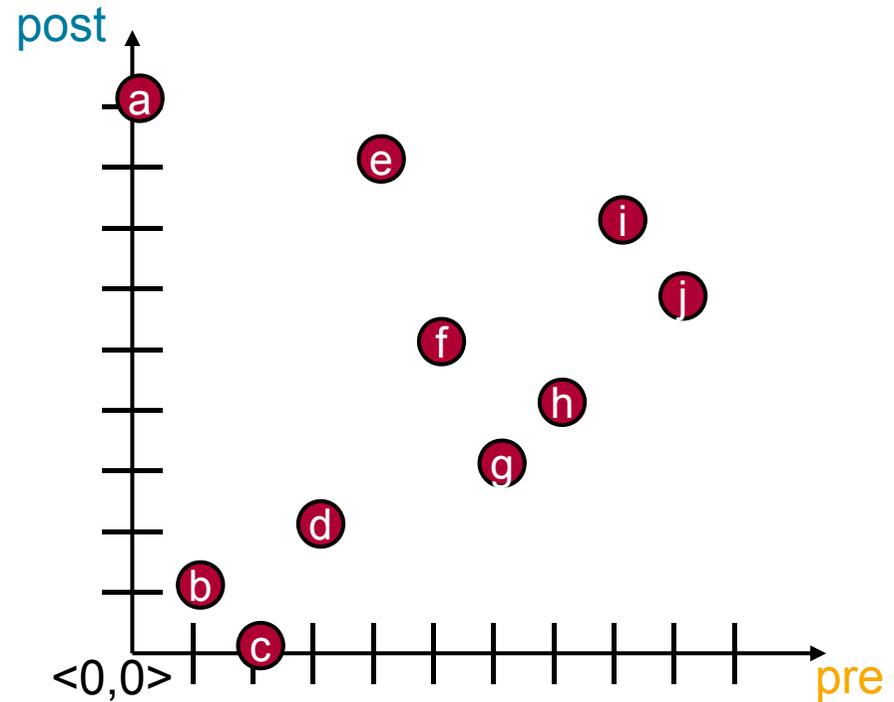
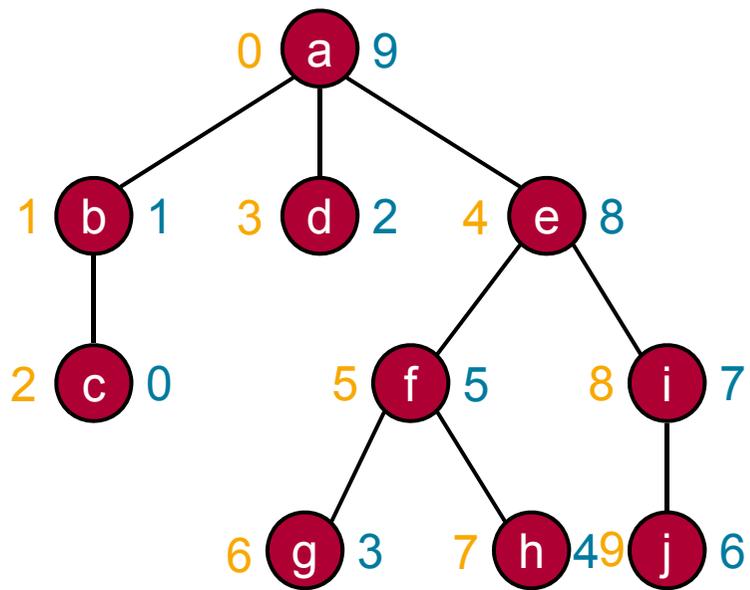
## ■ Postorder

- Knoten erhält postorder rank nach seinen Kindern.



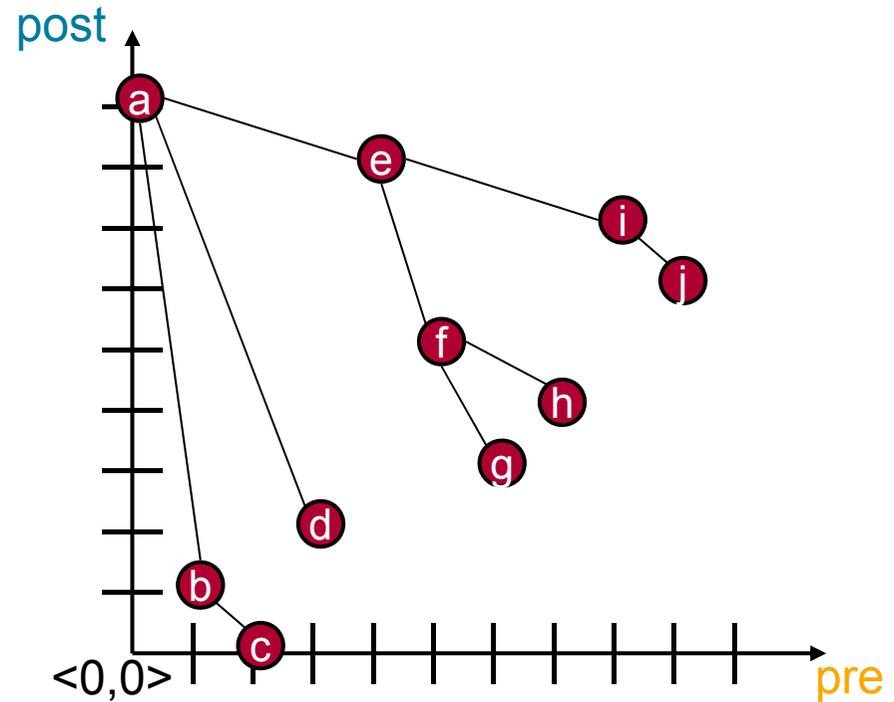
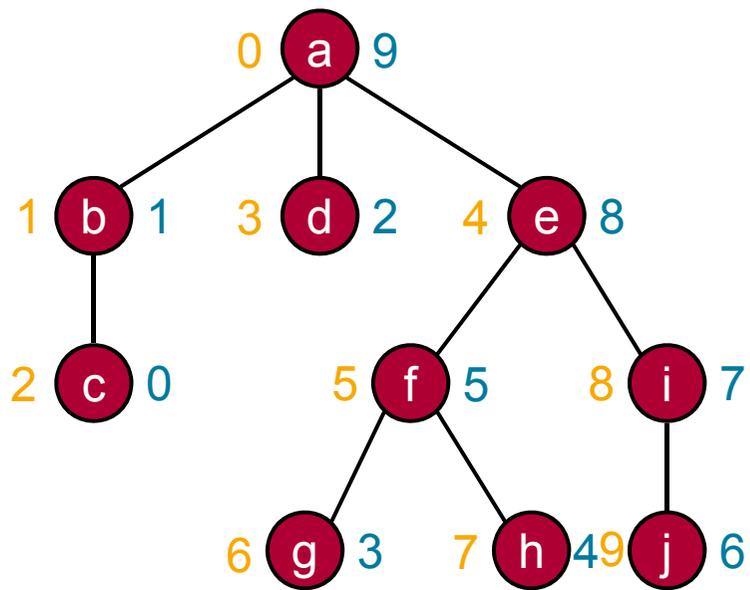
# Pre- post-order im Koordinatensystem

53



# Pre- post-order im Koordinatensystem

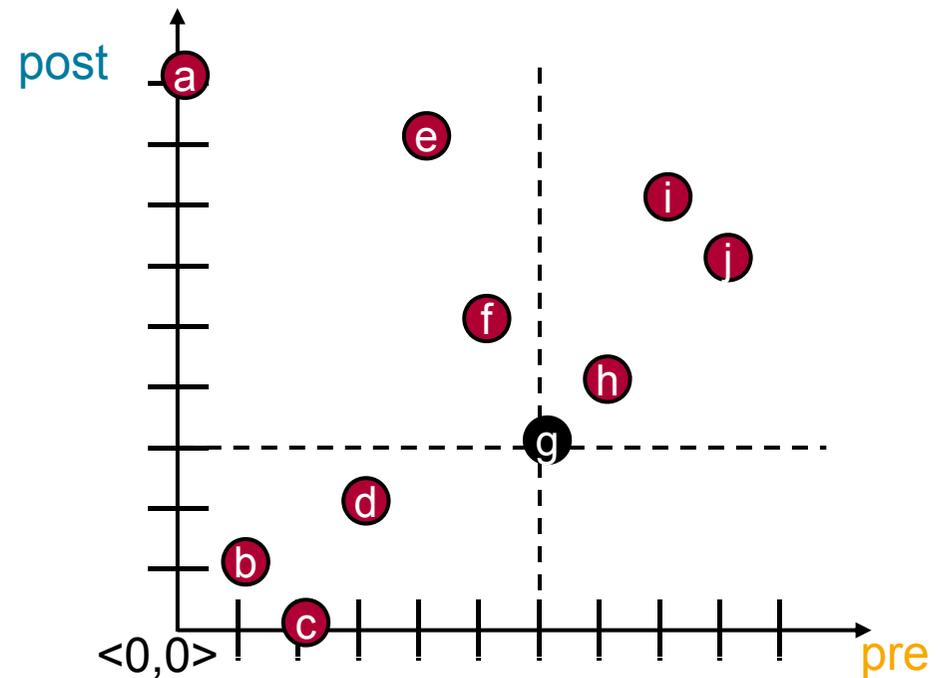
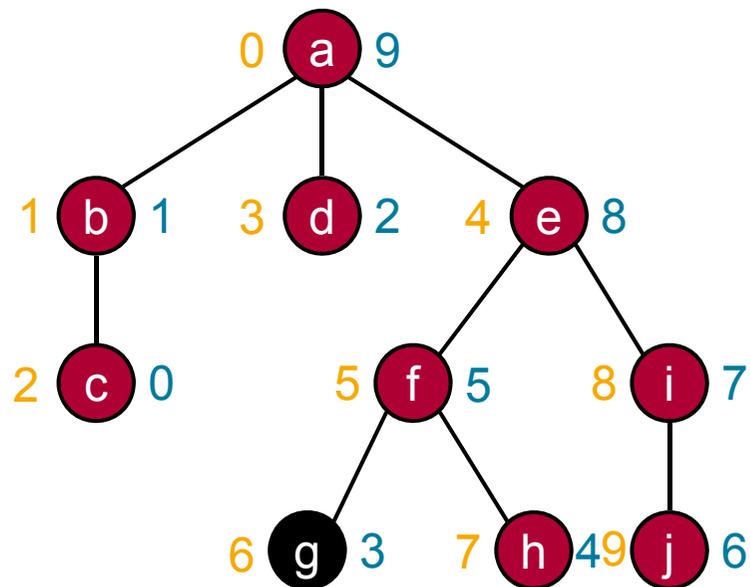
54



# Pre- post-order im Koordinatensystem

55

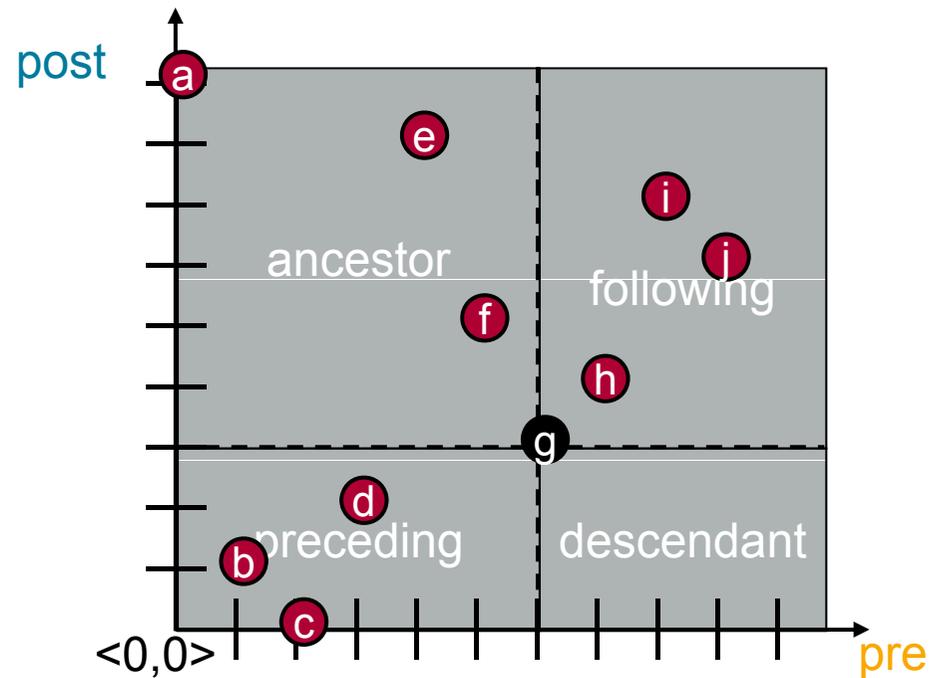
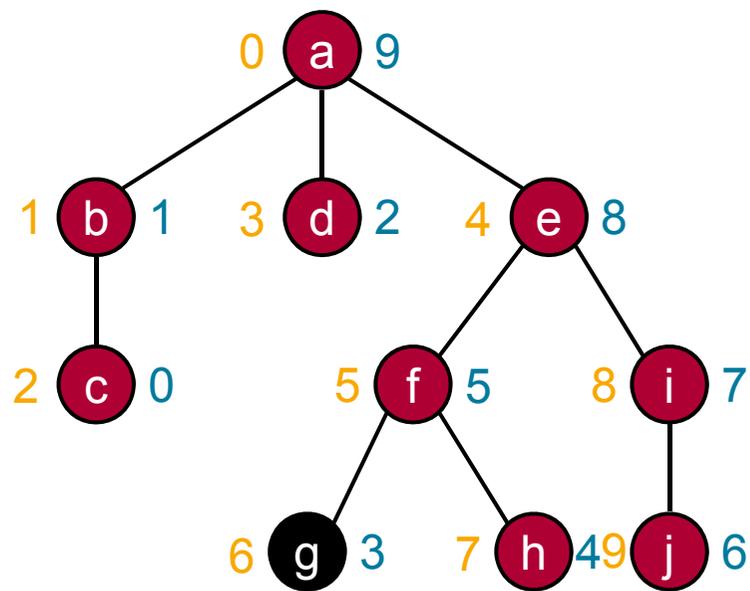
- Jeder Knoten (Kontextknoten) teilt die Ebene in vier Partitionen.



# Pre- post-order im Koordinatensystem

56

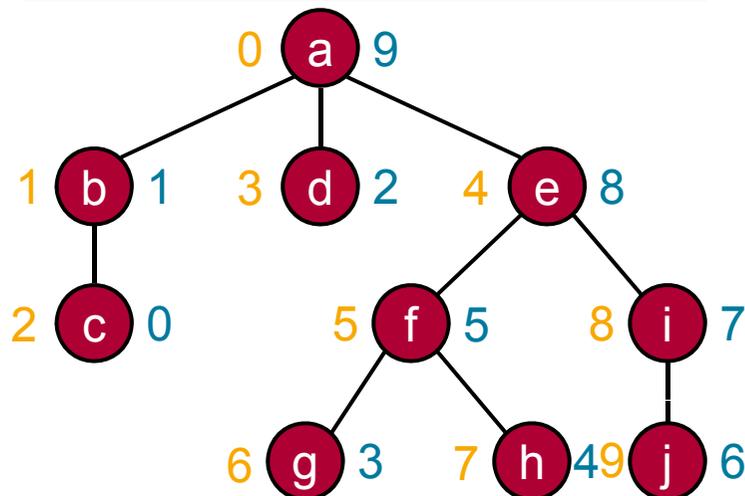
- Verfolgung eines Achs-Schrittes in einer XPath Anfrage entspricht einer rechteckigen Region der Ebene.



# Relationale Speicherung der Pre- Post-Order

```

<a>
  <b>c</b>
  <?d?>
  <e>
    <f><!--g-->h</f>
    <i>j</i>
  </e>
</a>
    
```



pre	post
0	9
1	1
2	0
3	2
4	8
5	5
6	3
7	4
8	7
9	6

pre	tag
0	a
1	b
4	e
5	f
8	i

pre	text
2	„c“
7	„h“
9	„j“

- Erstellung der Tabellen in einem einzelnen Scan
- Wiederherstellung des XML Dokuments möglich
- Weitere Tabelle für **types** möglich.

# XPath-2-SQL

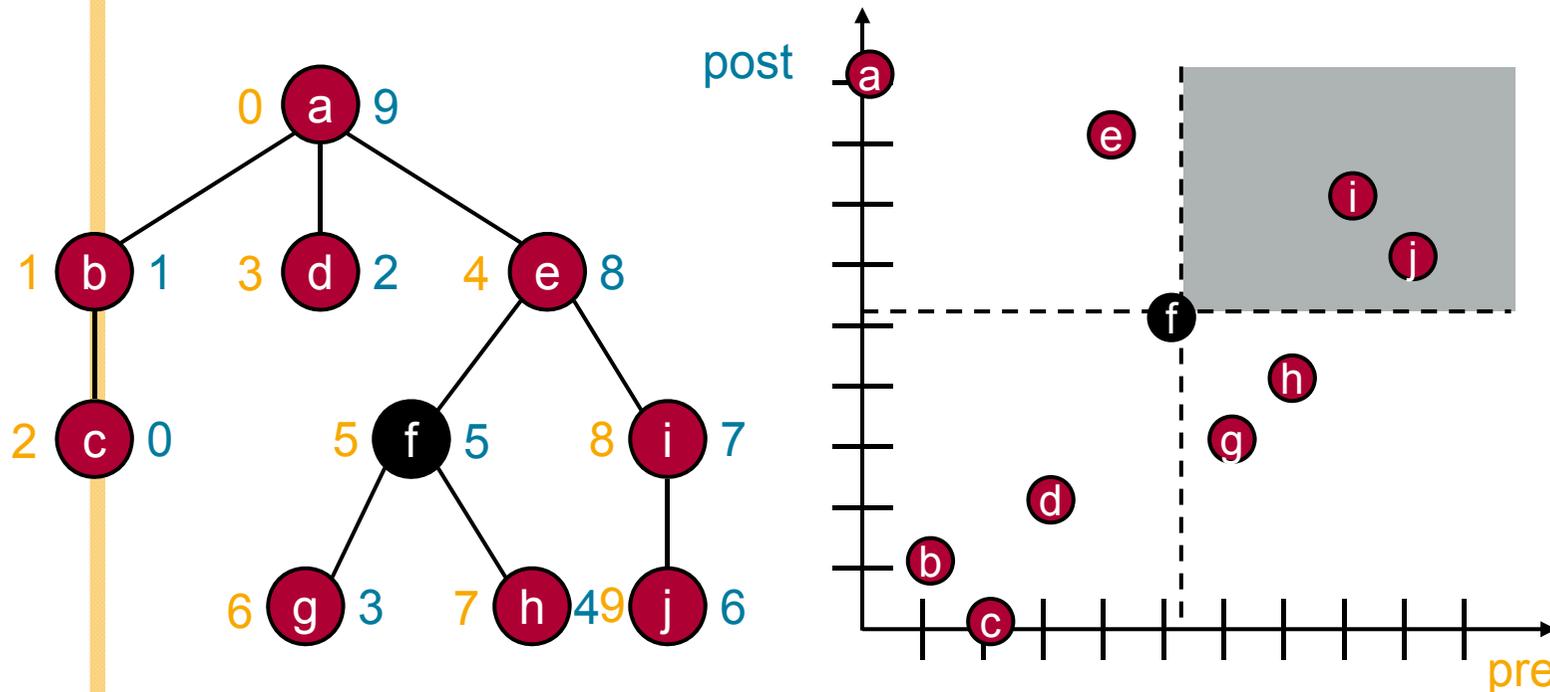
f/following

```

SELECT      DISTINCT P.pre
FROM        context C, prepost P
WHERE       P.pre > C.pre
AND         P.post > C.post
ORDER BY   P.pre
    
```

prepost

pre	post
0	9
1	1
2	0
3	2
4	8
5	5
6	3
7	4
8	7
9	6



# XPath-2-SQL

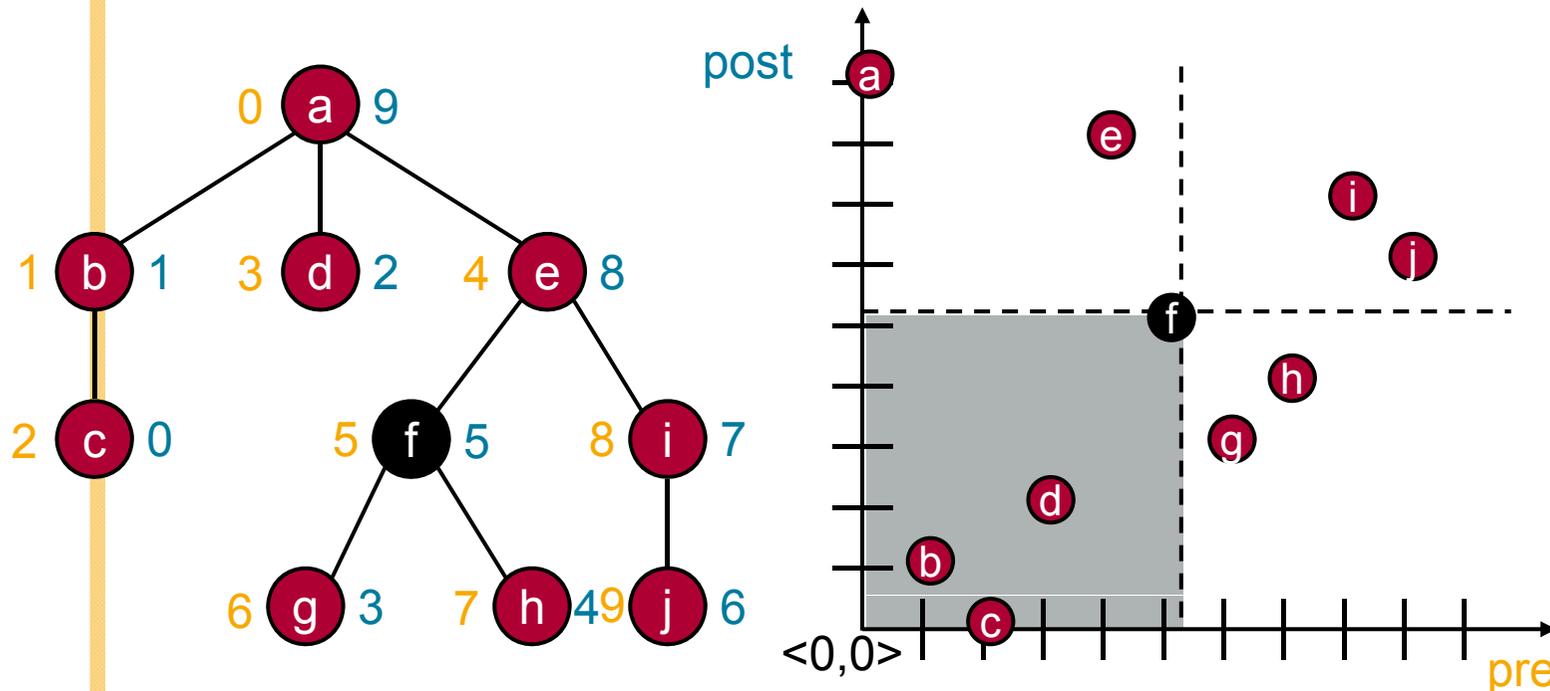
f/preceding

```

SELECT      DISTINCT P.pre
FROM        context C, prepost P
WHERE       P.pre < C.pre
AND         P.post < C.post
ORDER BY   P.pre
    
```

prepost

pre	post
0	9
1	1
2	0
3	2
4	8
5	5
6	3
7	4
8	7
9	6



# Baum-Bewusstsein

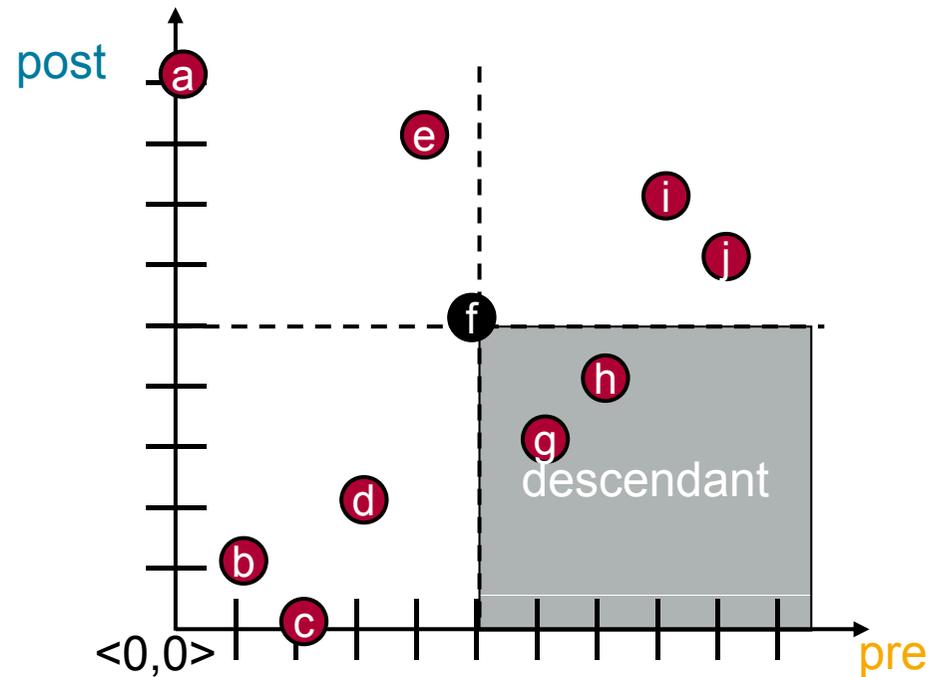
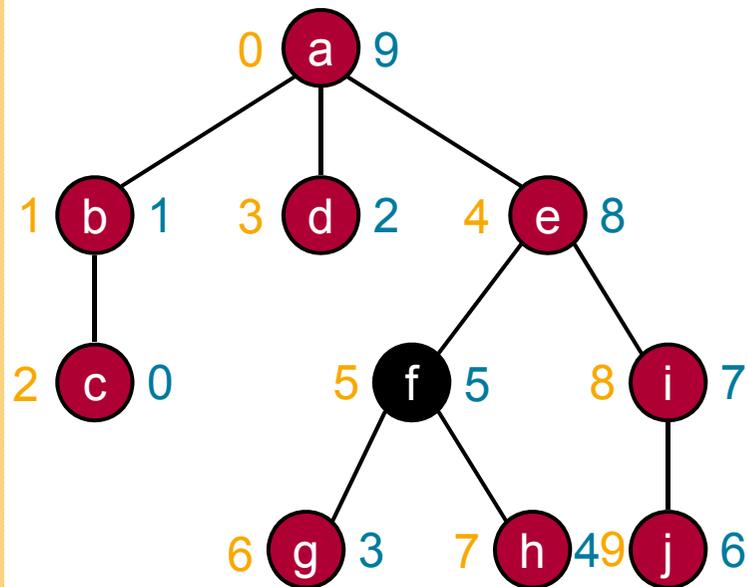
60



# Descendant-Bewusstsein

61

- Gesucht: f/descendant
- Prädikate
  - ... WHERE v\*.pre > f.pre AND v\*.post < f.post



# Index-Scans zur Anfragebearbeitung

## Beispielanfrage

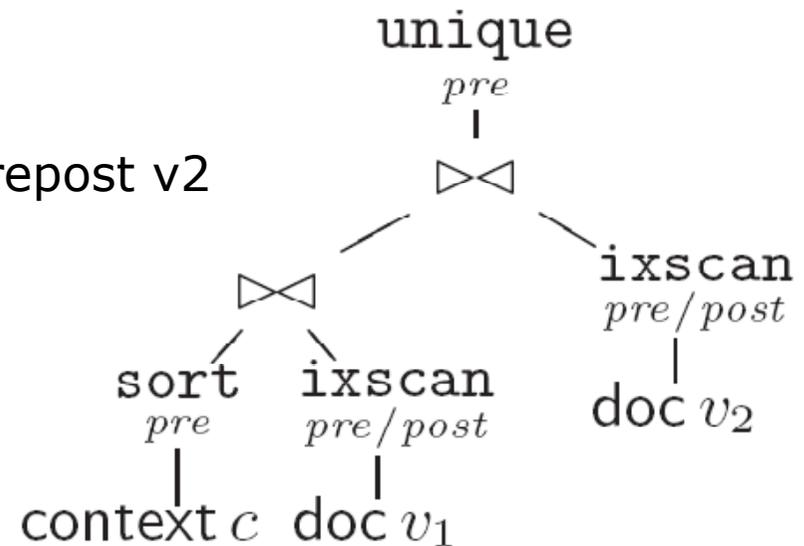
- context/following::node()/descendant::node()
- In Worten: Alle Knoten, die Nachfolger meiner folgenden Geschwister sind
- Anfrage

```

□ SELECT DISTINCT v2.pre
  FROM context c,prepost v1,prepost v2
 WHERE  v1.pre > c.pre
        AND v1.pre < v2.pre
        AND v1.post > c.post
        AND v1.post > v2.post
 ORDER BY v2.pre
  
```

## Ausführung im DBMS

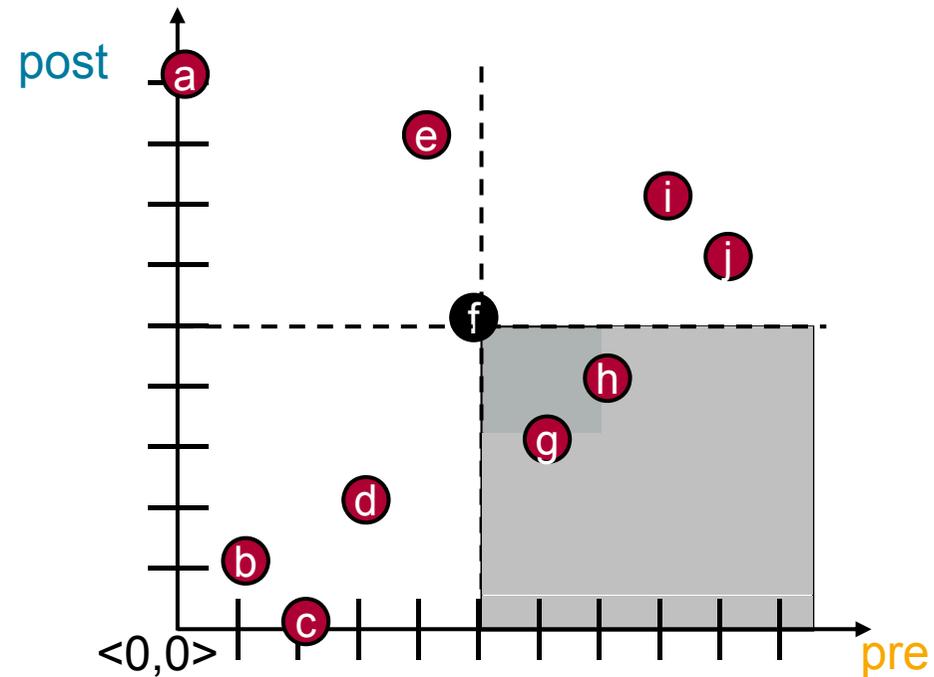
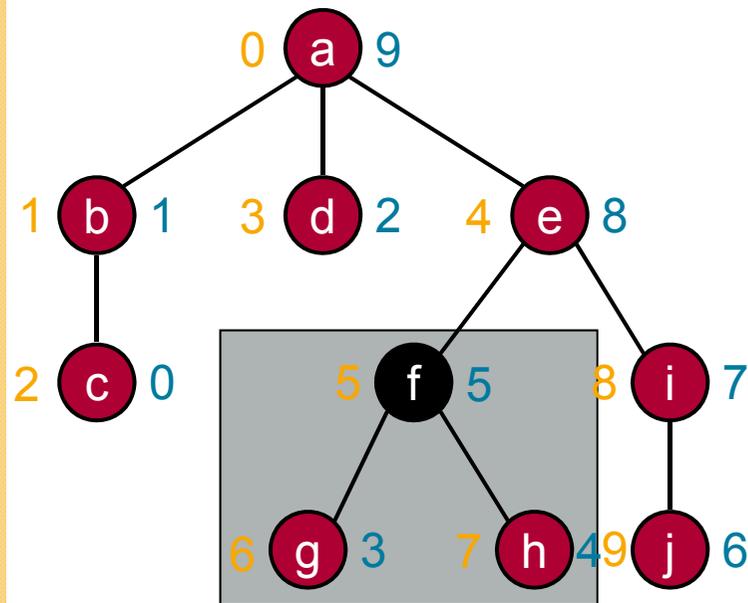
- Effizienz gut
- Aber jeweils gesamter Index wird gescannt.



# descendant-pruning

63

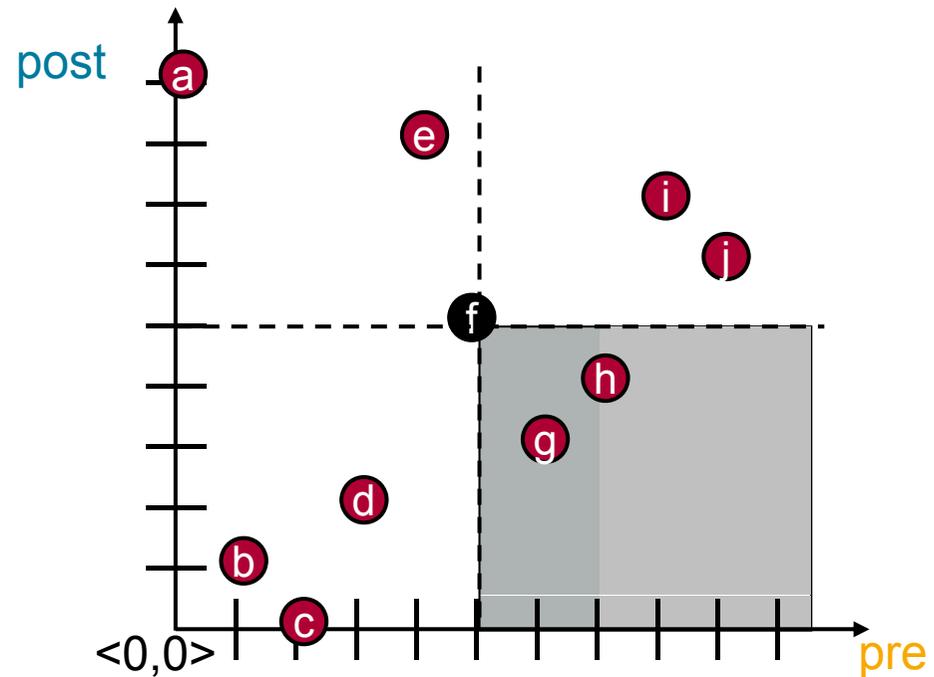
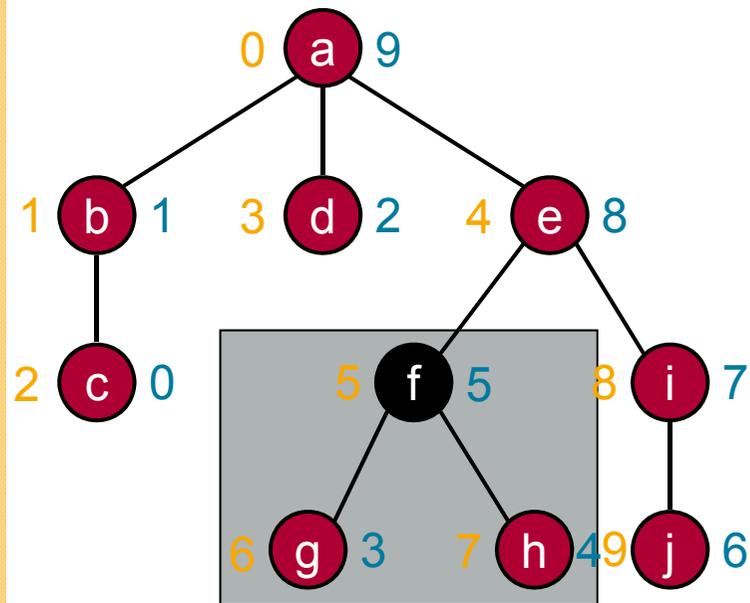
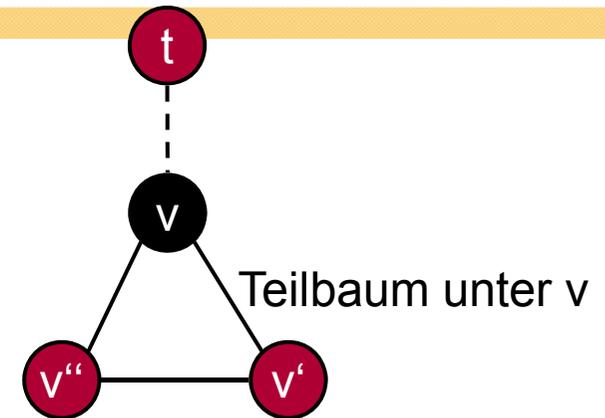
- ... WHERE  $v^*.pre > f.pre$  AND  $v^*.post < f.post$
- Aber:
  - Maximaler pre-Wert im Teilbaum unter f?
    - ◇  $\Rightarrow$  Ab  $pre > 7$  ist der Scan unnötig.
  - Minimaler post-Wert im Teilbaum unter f?
    - ◇  $\Rightarrow$  Ab  $post < 3$  ist der Scan unnötig.
- Problem: Wie nutze ich dieses Baum-Wissen aus?



# Pruning in der pre-Achse

64

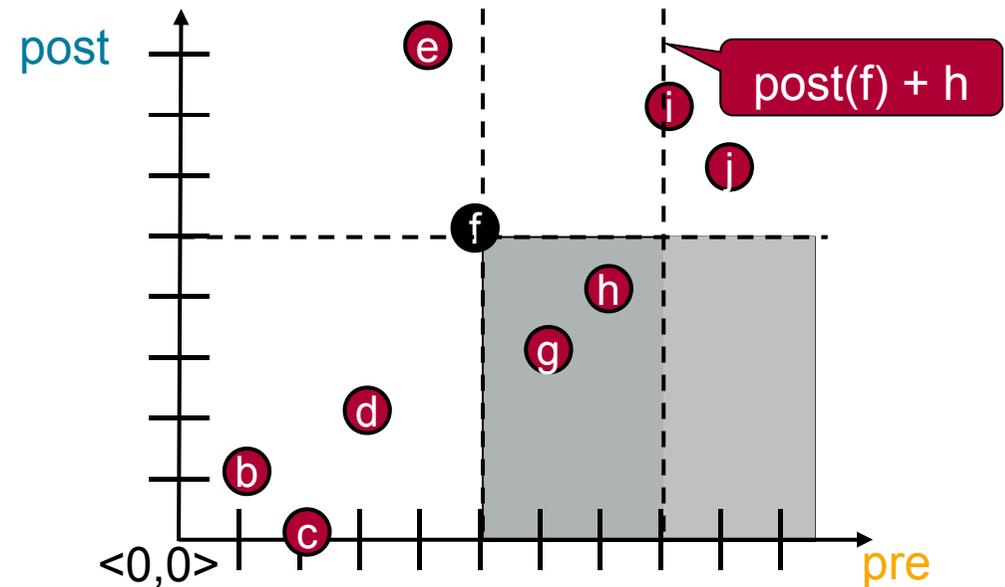
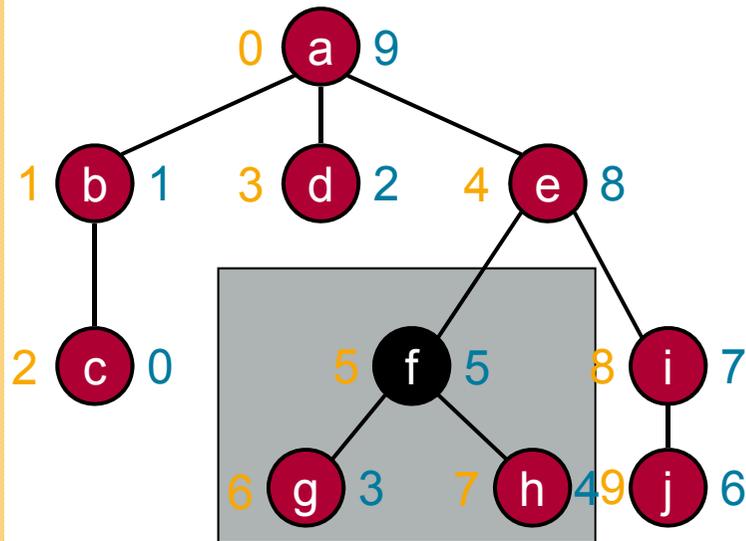
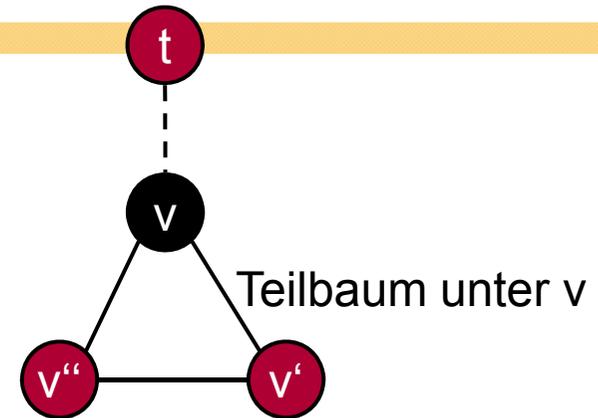
- Allgemein:
  - $v$ /descendant = Teilbaum unter  $v$
- Erkenntnis
  - Es reicht ein Scan von  $pre(v)$  bis  $pre(v')$
- Problem
  - Ich kenne  $pre(v')$  nicht ☹



# Berechnung $pre(v')$

65

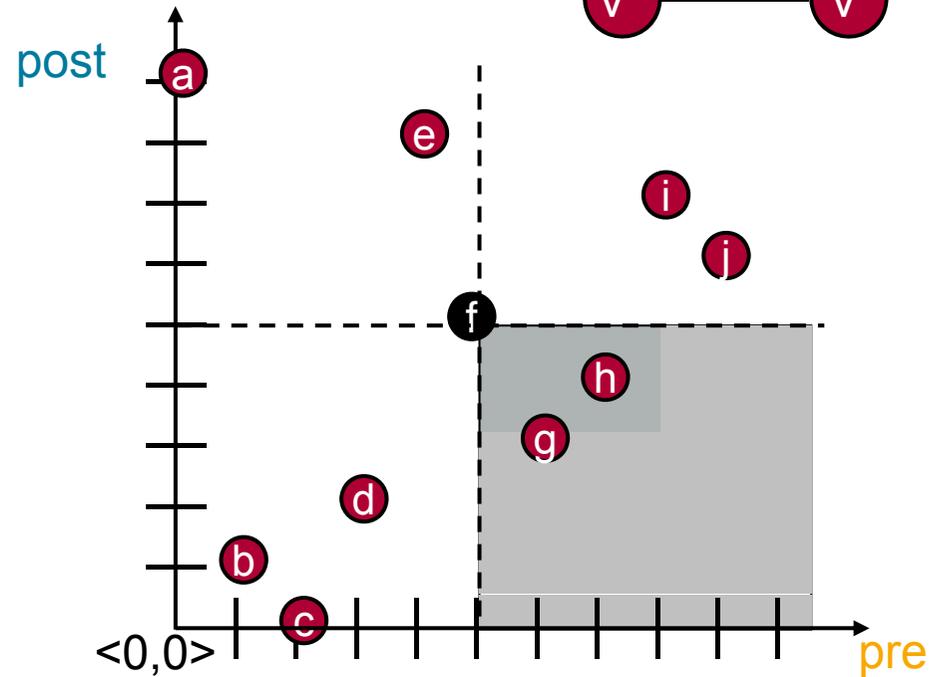
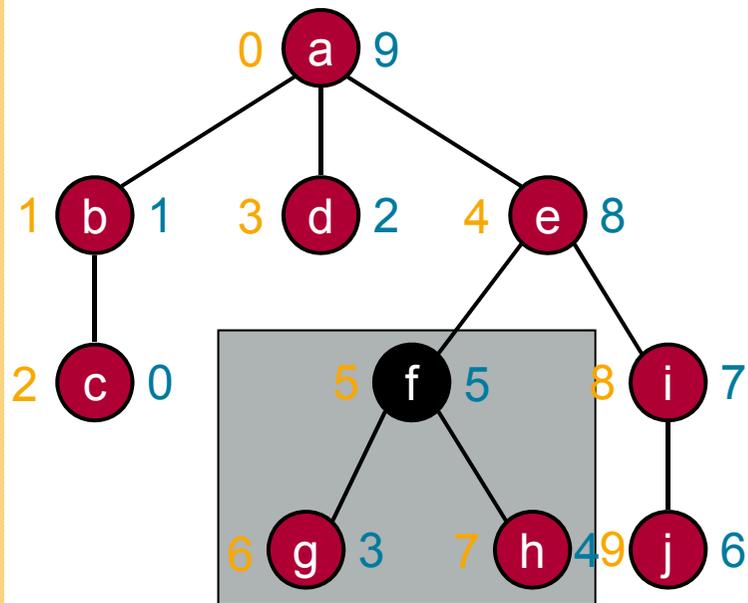
- $pre(v') = pre(v) + |v/\text{descendant}|$ 
  - Bsp:  $pre(h) = pre(f) + 2 = 7$
- $|v/\text{descendant}| = post(v) - pre(v) + level(v)$ 
  - $= 5 - 5 + 2 = 2$
- $level(v) \leq h = \text{Höhe des Gesamtbaums}$ 
  - $level(f) = 2 \leq 3$
- Zusammen:  $pre(v') \leq pre(v) + post(v) - pre(v) + h = post(v) + h$



# Pruning in der post-Achse

66

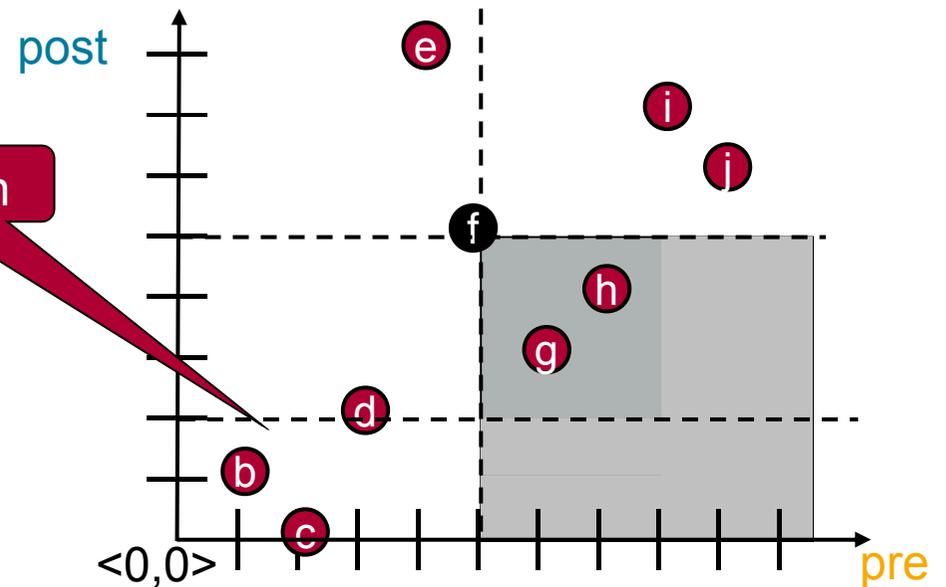
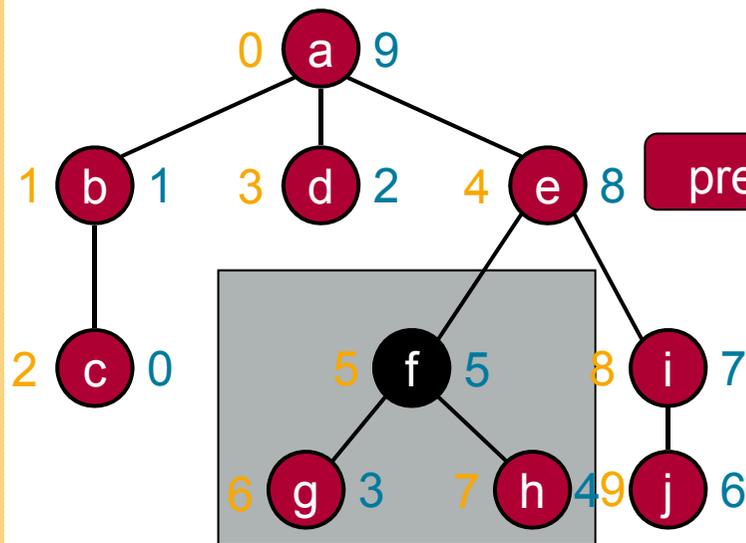
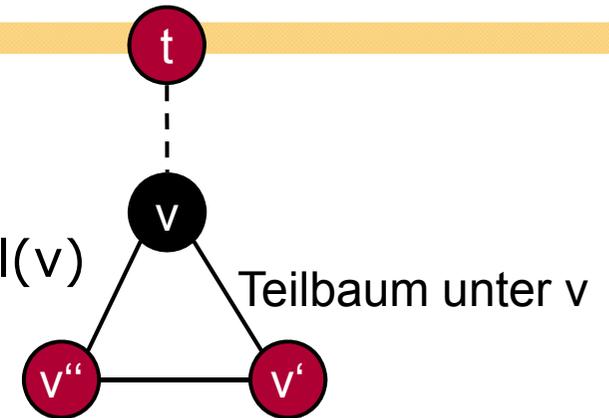
- Erkenntnis
  - Es reicht ein Scan von  $post(v'')$  bis  $post(v)$
- Problem
  - Ich kenne  $post(v')$  nicht ☹



# Berechnung $post(v'')$

67

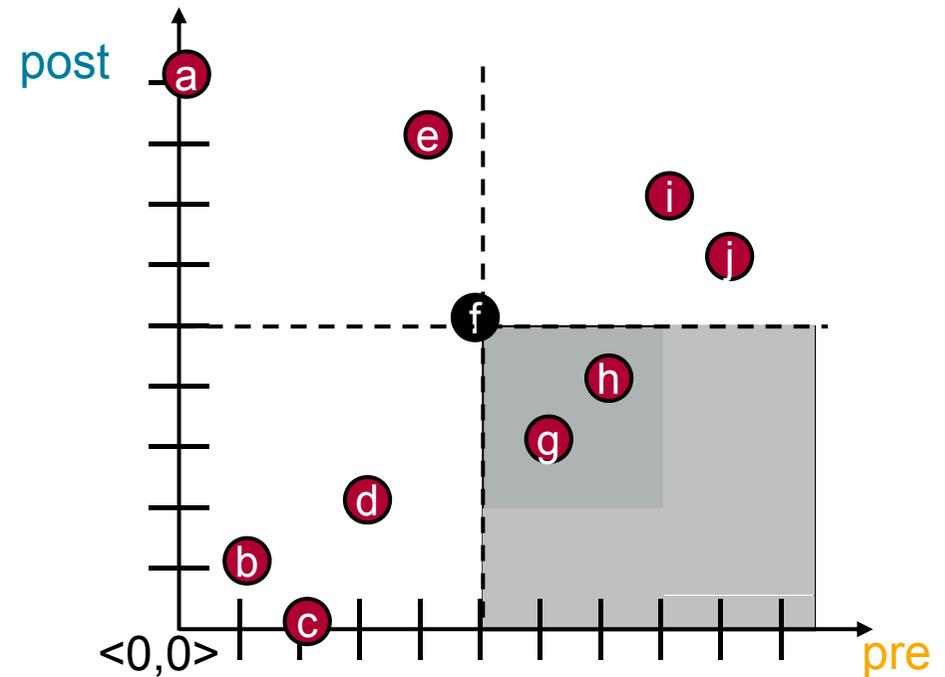
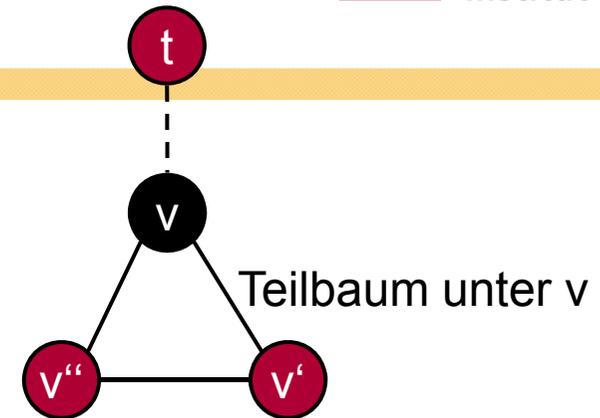
- $post(v'') = post(v) - |v/\text{descendant}|$ 
  - Bsp:  $post(g) = post(f) - 2 = 3$
- $|v/\text{descendant}| = post(v) - pre(v) + level(v)$ 
  - $= 5 - 5 + 2 = 2$
- $level(v) \leq h = \text{Höhe des Gesamtbaums}$ 
  - $level(f) = 2 \leq 3$
- Zusammen:  $post(v'') \geq post(v) - (post(v) - pre(v) + h) = pre(v) - h$



# Descendant-pruning

68

- Insgesamt
  - Vorher
  - ...
  - WHERE  $v^*.pre > f.pre$
  - AND  $v^*.post < f.post$
  - Nachher
  - ...
  - WHERE  $v^*.pre > f.pre$
  - AND  $v^*.pre \leq f.post + h$
  - AND  $v^*.post \geq f.pre + h$
  - AND  $v^*.post < f.post$
- Größe von h
  - Typischerweise  $< 10$
  - Verglichen mit Millionen Knoten



# Pruning allgemein

69

- Ähnliche Ideen für jede Achse
- Gescannter Indexbereich erheblich eingeschränkt
  
- Bisher: Pruning für einzelne Kontextknoten
- Aber: Typische XPath-Schritte haben viele Knoten als Input
  - `context/following::node()/descendant::node()`
- Deshalb: Pruning für Mengen von Knoten
  - Ein andermal...