Hasso Plattner Institut

IT Systems Engineering | Universität Potsdam

Datenbanksysteme I

XML & Datenbanken

6.7.2011

Felix Naumann

Überblick

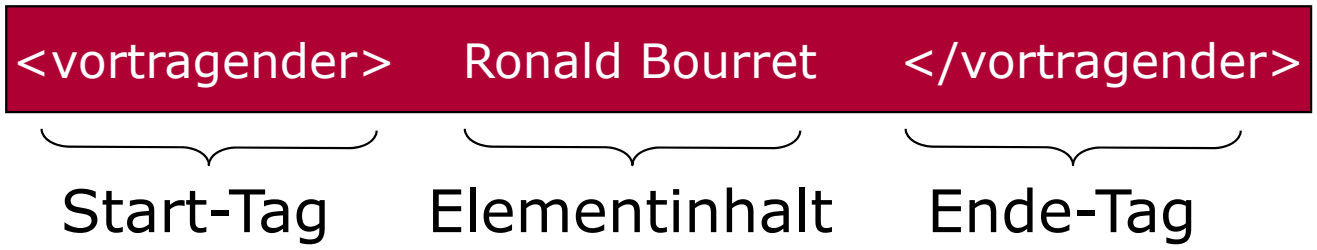
- Motivation & Syntax
- XML Programmierung
- Schemata
- Anfragesprachen
- Speicherung von XML

Motivation

- 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

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



Syntax von Elementen

- Leere Elemente:

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

- Schachtelung:

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

Start-Tag

Elementinhalt

Ende-Tag

Klassifikation von XML-Dokumenten

- 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


```xml
<?xml version="1.0" encoding="UTF-8"?>
<rechnung kundennummer="k333063143">
    <monatspreis>0,00</monatspreis>
    <einzelverbindungsnachweis>
        <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>
    </einzelverbindungsnachweis>
</rechnung>
```

XML-Dokument, Eigenschaften

- 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

- 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

Überblick

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

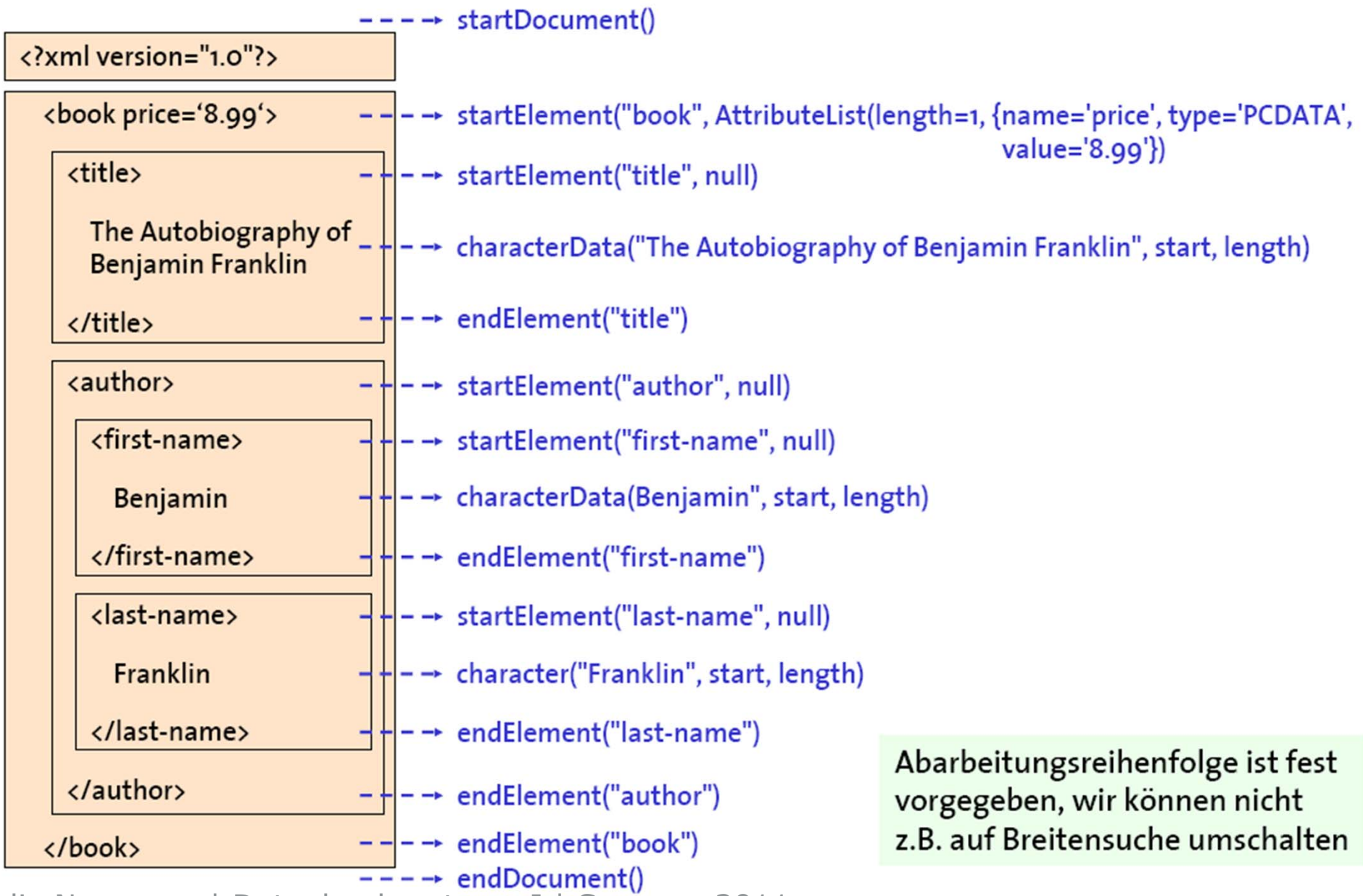
XML-Prozessoren

- 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

- 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

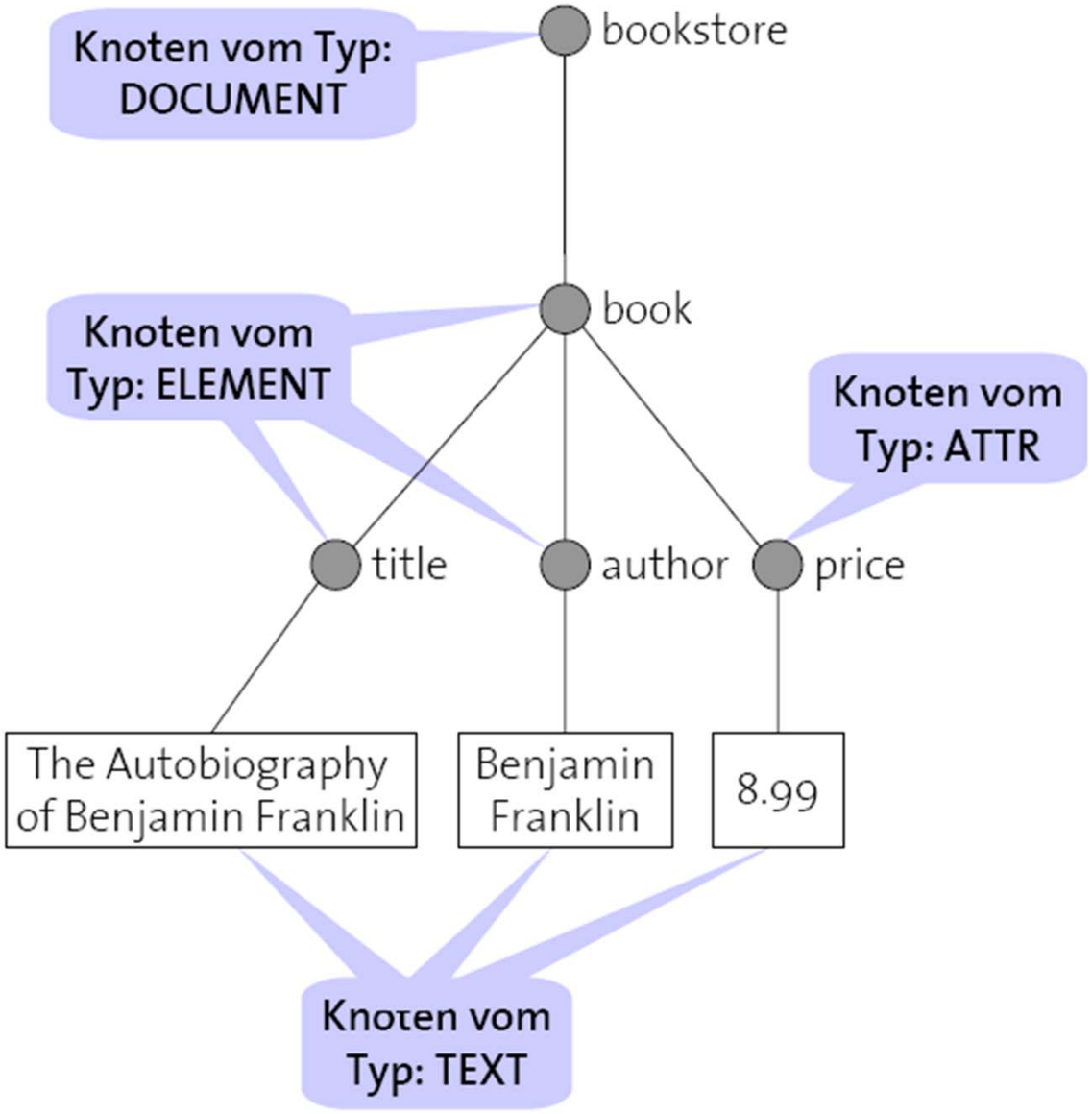
DOM – Document Object Model

- 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

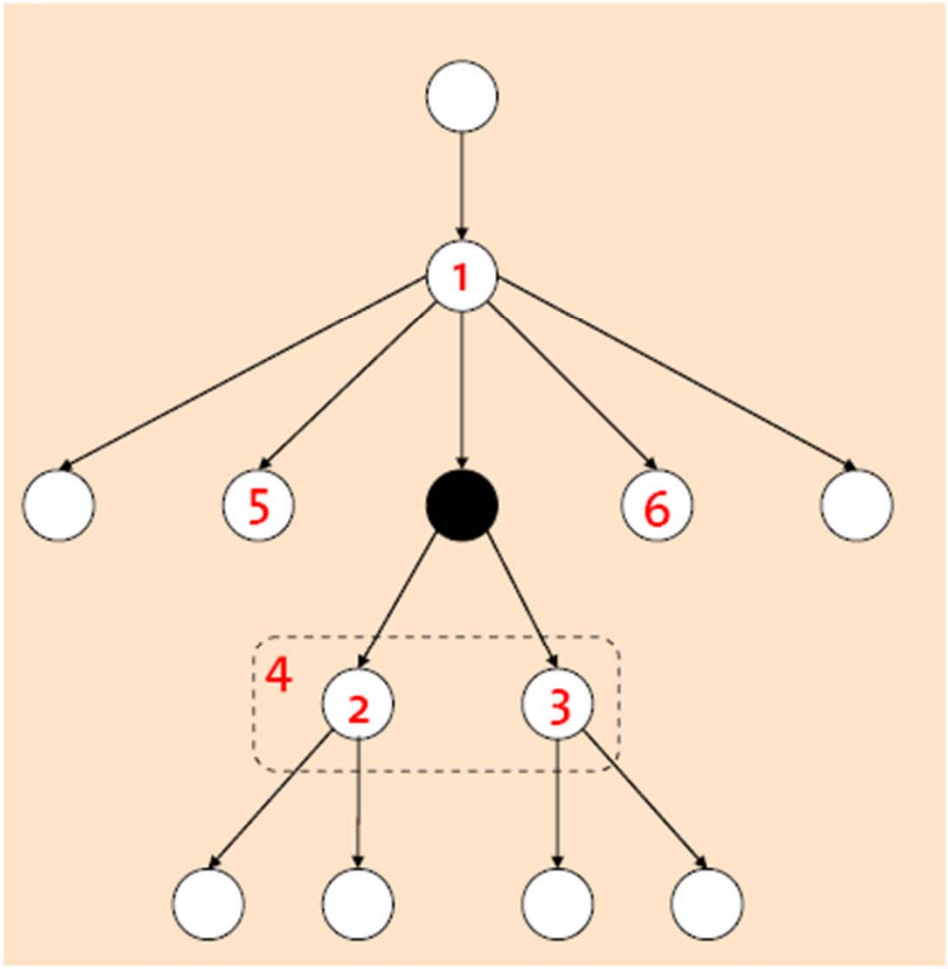
Quelle: Can Türker

DOM – als Baum


```
<?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



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

1 - getParentNode()

2 - getFirstChild()

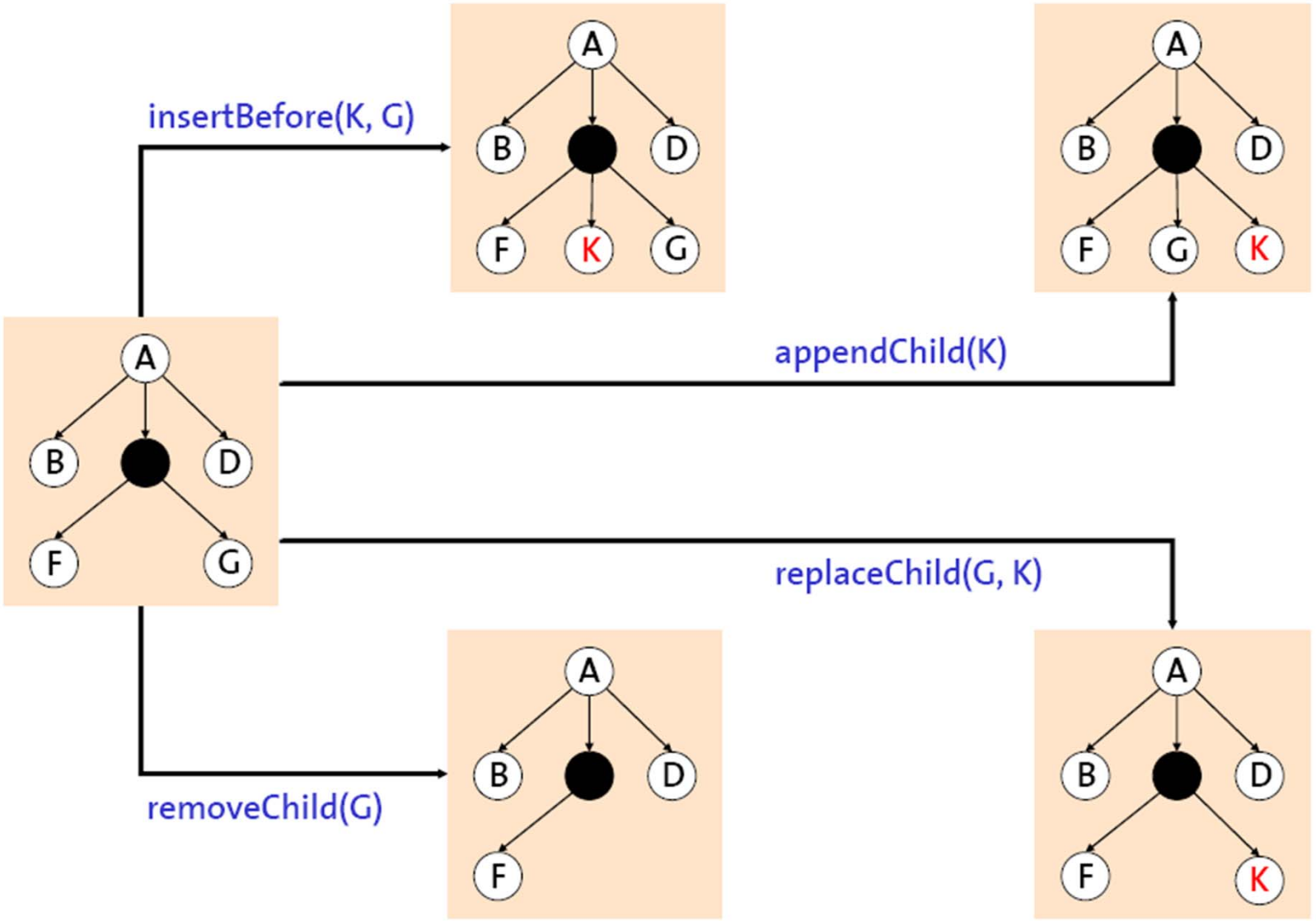
3 - getLastChild()

4 - getChildren()

5 - getPreviousSibling()

6 - getNextSibling()

DOM – Manipulation der Struktur



Vergleich von SAX und DOM

- 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
    - Anmerkung: XML Kompression erreicht ebenfalls Faktor 10

Überblick

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

DTDs


```
<?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>
```

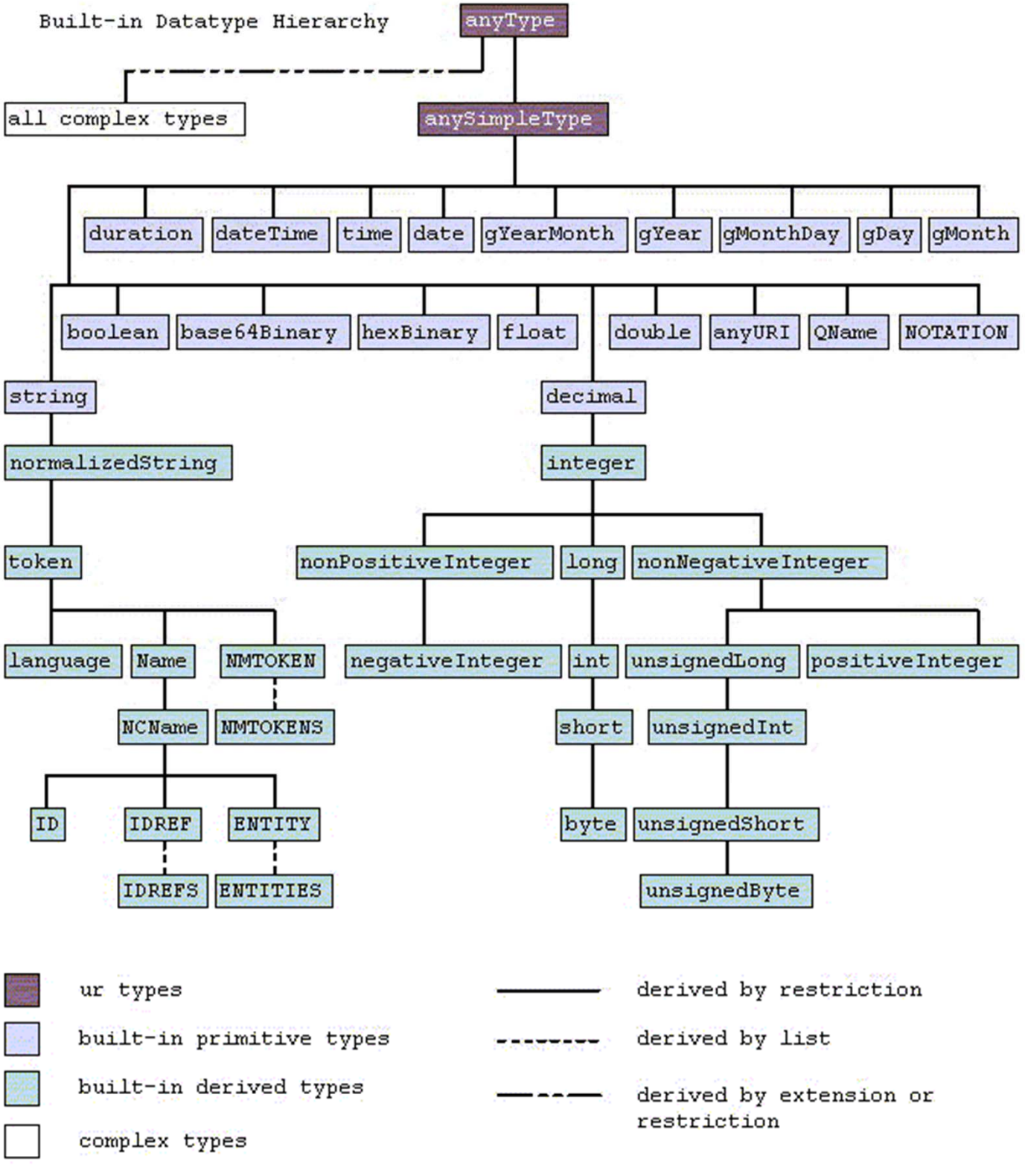
```
<!-- Hotel DTD-->
<!ELEMENT hotel (name, kategorie?,
adresse, hausbeschreibung, preise*)>
<!ATTLIST hotel id ID #REQUIRED
      url CDATA #REQUIRED>
<!ELEMENT  name (#PCDATA)>
<!ELEMENT  kategorie (#PCDATA)>
<!ELEMENT  adresse (plz, ort, strasse,
hausnummer, telefon, fax?, e-mail?)>
<!ELEMENT  plz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

XML Schema

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



XML Schema – Beispiel


```
<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>
```

Überblick

- Motivation & Syntax
- XML Programmierung
- Schemata
- Anfragesprachen
  - XPath, XQuery, XSLT
- Speicherung von XML

Anfragesprachen – Grundoperationen

- Selection
  - Choosing a document or document element based on content, structure or attributes.
- Extraction
  - Pulling out particular elements of a document.
- Reduction
  - Removing selected sub-elements of an element.
  - ≈ projection
- Restructuring
  - Constructing a new set of element instances to hold queried data.
- Combination
  - Merging two or more elements into one.

Anfragesprachen – Selektion & Extraction


```
<manufacturer>
 <mn-name>Mercury</mn-name>
 <year>1999</year>
 <model> <mo-name>Sable LT</mo-name>
   <front-rating>3.84</front-rating>
   <side-rating>2.14</side-rating>
   <rank>9</rank>
 </model>
 <model>
   …
 </model>
 …
</manufacturer>
```

```
<vehicle>
 <vendor>Scott Thomason</vendor>
 <make>Mercury</make>
 <model>Sable LT</model>
 <year>1999</year>
 <color>metallic blue</color>
 <option opt="sunroof"/>
 <option opt="A/C"/>
 <option opt="lthr seats"/>
 <price>26800</price>
</vehicle>
```

- Gegeben: 2 XML Dokumente.
- Anfrage: Finde alle Autos mit Angaben zu Hersteller, Modell, Verkäufer, Rang und Preis, die einen besseren Rang als 10 haben, aber weniger als 30000 kosten.

Anfragesprachen – Selektion & Extraction


```
<manufacturer>
 <mn-name>Mercury</mn-name>
 <year>1999</year>
 <model> <mo-name>Sable LT</mo-name>
   <front-rating>3.84</front-rating>
   <side-rating>2.14</side-rating>
   <rank>9</rank>
 </model>
 <model>
   …
 </model>
 …
</manufacturer>
```

```
<vehicle>
 <vendor>Scott Thomason</vendor>
 <make>Mercury</make>
 <model>Sable LT</model>
 <year>1999</year>
 <color>metallic blue</color>
 <option opt="sunroof"/>
 <option opt="A/C"/>
 <option opt="lthr seats"/>
 <price>26800</price>
</vehicle>
```

- Selektiere und extrahiere manufacturer, die mindestens ein model mit rank < 10 haben.
- Selektiere und extrahiere vehicles, die weniger als 30000 kosten.

Anfragesprachen – Reduction


```
<manufacturer>
 <mn-name>Mercury</mn-name>
 <year>1999</year>
 <model> <mo-name>Sable LT</mo-name>
   <front-rating>3.84</front-rating>
   <side-rating>2.14</side-rating>
   <rank>9</rank>
 </model>
 <model>
   …
 </model>
 …
</manufacturer>
```

```
<vehicle>
 <vendor>Scott Thomason</vendor>
 <make>Mercury</make>
 <model>Sable LT</model>
 <year>1999</year>
 <color>metallic blue</color>
 <option opt="sunroof"/>
 <option opt="A/C"/>
 <option opt="lthr seats"/>
 <price>26800</price>
</vehicle>
```

- Eliminiere aus den verbliebenen manufacturer diejenigen model, die rank < 10 haben.
- Eliminiere front-rating und side-rating.
- Eliminiere color und option Elemente aus allen vehicle.

Anfragesprachen – Combination


```
<manufacturer>
 <mn-name>Mercury</mn-name>
 <year>1999</year>
 <model>
   <mo-name>Sable LT</mo-name>
   <rank>9</rank>
 </model>
</manufacturer>
```

```
<vehicle>
 <vendor>Scott Thomason</vendor>
 <make>Mercury</make>
 <model>Sable LT</model>
 <year>1999</year>
 <price>26800</price>
</vehicle>
```

- Kombiniere manufacturer und vehicle wobei mo-name = model und mn-name = make.

Anfragesprachen – Restructuring

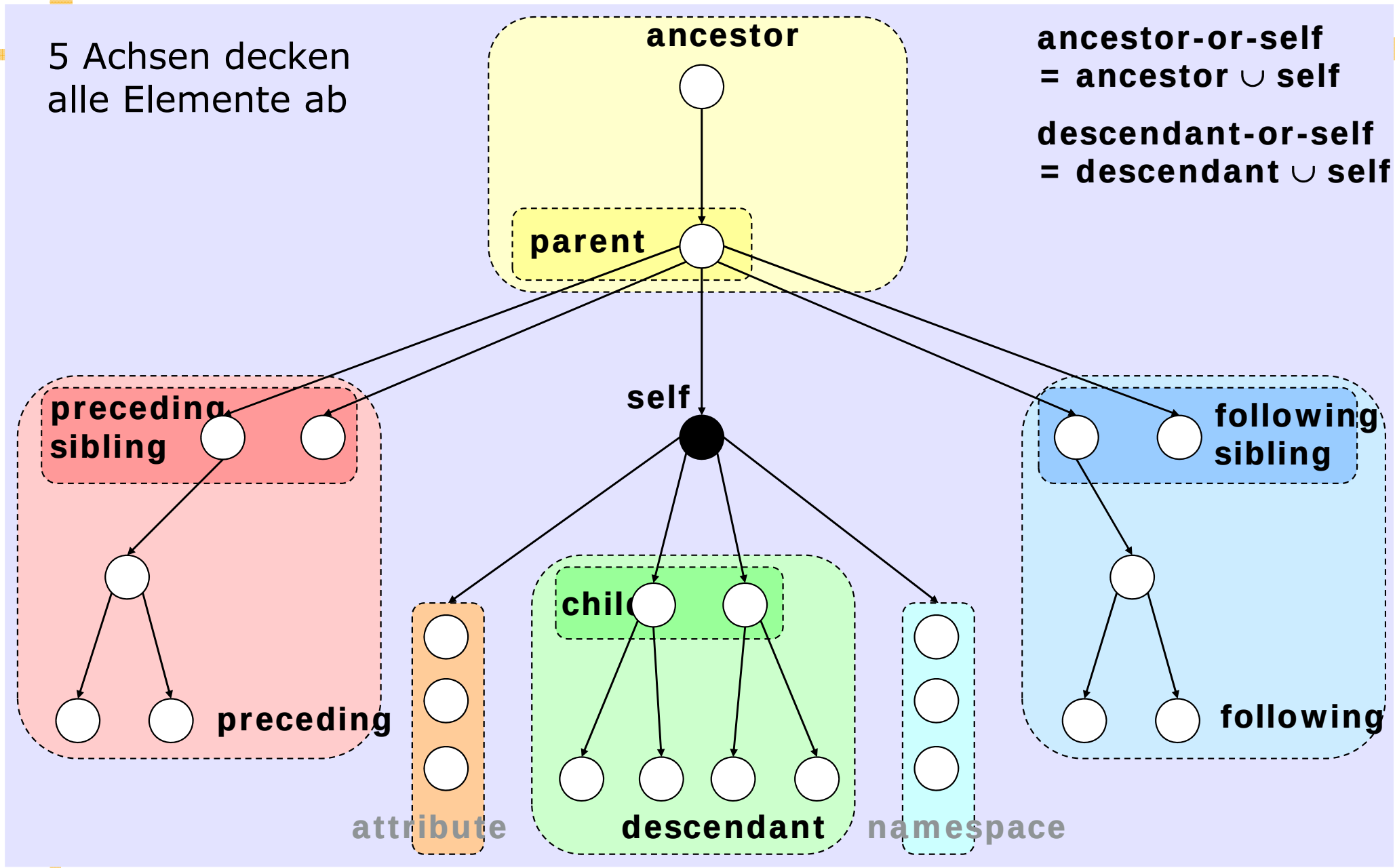

```
<manufacturer>
 <mn-name>Mercury</mn-name>
 <year>1999</year>
 <model>
   <mo-name>Sable LT</mo-name>
   <rank>9</rank>
 </model>
</manufacturer>
```

```
<vehicle>
 <vendor>Scott Thomason</vendor>
 <make>Mercury</make>
 <model>Sable LT</model>
 <year>1999</year>
 <price>26800</price>
</vehicle>
```

- Strukturiere die Kombination so, dass das Ergebnis die folgende Form hat:

```
<car>
 <make>Mercury</make>
 <model>Sable LT</model>
 <vendor>Scott Thomason</vendor>
 <rank>9</rank>
 <price>26800</rank>
</car>
```

XPath – Navigationsachsen



Syntax: Abkürzungen

Abkürzung	Langform	liefert
tagname	child::tagname	alle Kinderknoten, die 'tagname'-Elemente sind
.	self::node()	den aktuellen Knoten
..	parent::node()	den Elternknoten
*	descendant-or-self::	alle Nachkommen des aktuellen Knotens
@name	attribute::name	Attribut 'name' des aktuellen Knotens
/		den Wurzelknoten
//		alle Nachkommen des Wurzelknotens
[expr]		die Elemente aus der Knotenfolge, für die der aktuelle Teilpfad gilt und der Ausdruck true wird
[n]		das n-te Element aus der Knotenfolge, für die der aktuelle Teilpfad gilt

XPath-Funktionsbibliothek

Funktion	liefert
number last()	Position des letzten Elementes
number position()	Kontextposition
number sum(node-set)	Summe der zu Zahlen umgewandelten Argumentknoten
number count(node-set)	Anzahl der Argumentknoten
string name(node-set?)	Name des Argumentknotens
node-set id(object)	liefert den Knoten mit der ID, Auflösung von IDREF
boolean contains(string, string)	true, wenn zweites Argument Teil des ersten ist
boolean not(boolean)	Negation des angegebenen Wertes
…	…

Beispiele für XPath-Anfragen

- **`/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

Überblick über XQuery

- aktueller Vorschlag W3C für XML-Anfragesprache
  - Last Call
- 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 (ordered nodeset, Sequenz)
  - Flach
  - Geordnet (oder unordered())
  - Nicht duplikatfrei

Basiskonstrukte

- **Elementkonstruktoren** zur Erstellung oder Ableitung neuer XML-Elemente
- **Pfadausdrücke** (XPath) zur Selektion von Dokumentbestandteilen
- Anwendung von datentypspezischen **Operatoren**
- **Funktionsaufrufe**
  - neben Standardfunktionen nutzerdefinierte Funktionen
- **FLWR**-Ausdrücke (ähnlich SQL)
- **Bedingte** Ausdrücke (`if...then...else...`)
- Quantifizierte Ausdrücke
  - **Quantoren** `some, every`
- Test von **Datentypen**
- Weitere einfache Ausdrücke
  - Siehe folgende Folien

XQuery

- 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


```
<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


```
<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>
```

{ } kennzeichnet einen auszuwertenden Ausdruck

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

Ein **hotel**-Element pro **zimmertyp**

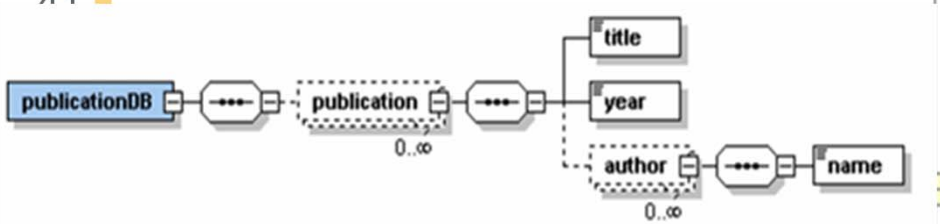
Beispiele


```
<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



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

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**

XQueryX

- Darstellung von XQuery Anfragen als XML
- Verschiedene syntax-bindings für XQuery
  - Human-readable (s.o.)
  - XML-Processor parsable
- Vorteile von XQueryX
  - Leicht zu parsen
  - Anfragen über Anfragen
    - Z.B.: Welche der 100 Anfragen sortiert?
  - Automatische Generierung von Anfragen
  - Embedding von Anfragen in XML Dokumente
- Definiert durch ein XML Schema
- XSLT zur Transformation XQueryX -> XQuery
- http://www.w3.org/TR/2005/WD-xqueryx-20050404/

XQueryX – X


```
<!-- The base class -
  <xsd:complexType na
  <xsd:element name="

  <!-- Simple wrapper
  <xsd:complexType na
    <xsd:sequence>
      <xsd:element re
    </xsd:sequence>
  </xsd:complexType>
```

```
<xsd:complexType name="flworExpr">
  <xsd:complexContent>
    <xsd:extension base="expr">
      <xsd:sequence>
        <xsd:choice maxOccurs="unbounded">
          <xsd:element ref="forClause"/>
          <xsd:element ref="letClause"/>
        </xsd:choice>
        <xsd:element name="whereClause" minOccurs="0"/>
        <xsd:element name="orderByClause" minOccurs="0"/>
        <xsd:element name="returnClause"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
<xsd:element name="forClauseItem">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="typedVariableBinding"/>
      <xsd:element ref="positionalVariableBinding" minOccurs="0"/>
      <xsd:element name="forExpr" type="exprWrapper"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
<xsd:element name="forClause">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="forClauseItem" maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
<xsd:element name="letClauseItem">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="typedVariableBinding"/>
      <xsd:element name="letExpr" type="exprWrapper"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

XQueryX

List books published by A

```
<bib>
 {
  for $b in doc("http://
  where $b/publisher = "
  return
    <book year="{ $b/@ye
     { $b/title }
    </book>
 }
</bib>
```



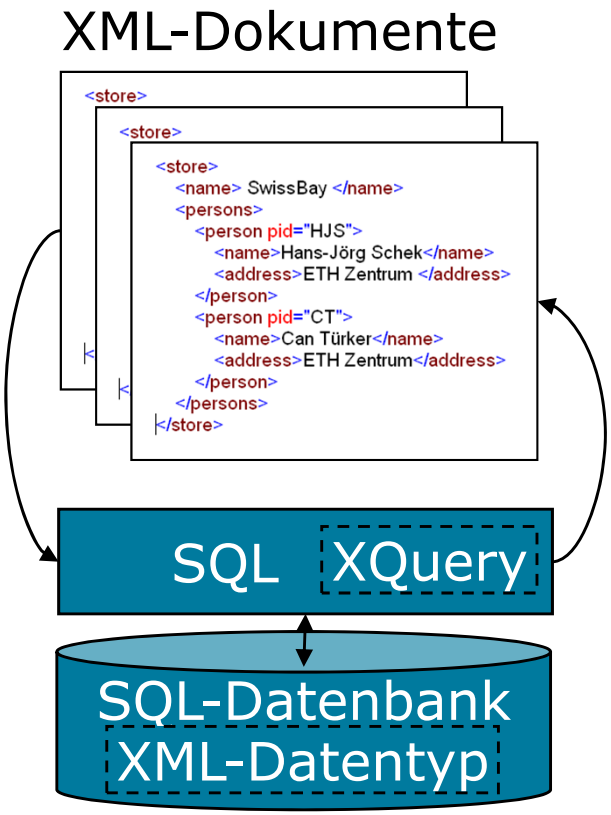
```
<?xml-stylesheet type="text/xsl" href="xqueryx.xsl" ?>
<xqx:module xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xqx="http://www.w3.org/2005/04/XQueryX" xsi:schemaLocation="
http://www.w3.org/2005/04/XQueryX/xqueryx.xsd">
  <xqx:mainModule>
    <xqx:queryBody>
      <xqx:expr xsi:type="xqx:elementConstructor">
        <xqx:tagName>bib</xqx:tagName>
        <xqx:elementContent>
          <xqx:expr xsi:type="xqx:flworExpr">
            <xqx:forClause>
              <xqx:forClauseItem>
                <xqx:typedVariableBinding>
                  <xqx:varName>b</xqx:varName>
                </xqx:typedVariableBinding>
                <xqx:forExpr>
                  <xqx:expr xsi:type="xqx:pathExpr">
                    <xqx:argExpr>
                      <xqx:expr xsi:type="xqx:functionCallExpr">
                        <xqx:functionName>doc</xqx:functionName>
                        <xqx:arguments>
                          <xqx:expr xsi:type="xqx:stringConstantExpr">
                            <xqx:value>http://bstore1.example.com/bib.xml</xqx:value>
                          </xqx:expr>
                        </xqx:arguments>
                      </xqx:expr>
                    </xqx:argExpr>
                    <xqx:stepExpr>
                      <xqx:xpathAxis>child</xqx:xpathAxis>
                      <xqx:nameTest>bib</xqx:nameTest>
                    </xqx:stepExpr>
                    <xqx:stepExpr>
                      <xqx:xpathAxis>child</xqx:xpathAxis>
                      <xqx:nameTest>book</xqx:nameTest>
                    </xqx:stepExpr>
                  </xqx:expr>
                </xqx:forExpr>
              </xqx:forClauseItem>
            </xqx:forClause>
            <xqx:whereClause>
              <xqx:expr xsi:type="xqx:operatorExpr">
                <xqx:infixOp/>
                <xqx:opType>and</xqx:opType>
                <xqx:arguments>
                  <xqx:expr xsi:type="xqx:operatorExpr">
                    <xqx:infixOp/>
                    <xqx:opType>=</xqx:opType>
                    <xqx:arguments>
                      <xqx:expr xsi:type="xqx:pathExpr">
                        <xqx:argExpr>
                          <xqx:expr xsi:type="xqx:varRef">
                            <xqx:name>b</xqx:name>
                          </xqx:expr>
                        </xqx:argExpr>
                        <xqx:stepExpr>
                          <xqx:xpathAxis>child</xqx:xpathAxis>
                          <xqx:nameTest>publisher</xqx:nameTest>
                        </xqx:stepExpr>
                      </xqx:expr>
```

Überblick

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

„SQL / XML“ – Grundidee

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



Speicherung von XML-Dokumenten in der Datenbank als Wert des XML-Datentyps

Generierung von XML-Dokumenten mittels SQL/XML-Funktionen

Basisdatentyp XML

- 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

- 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

- 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

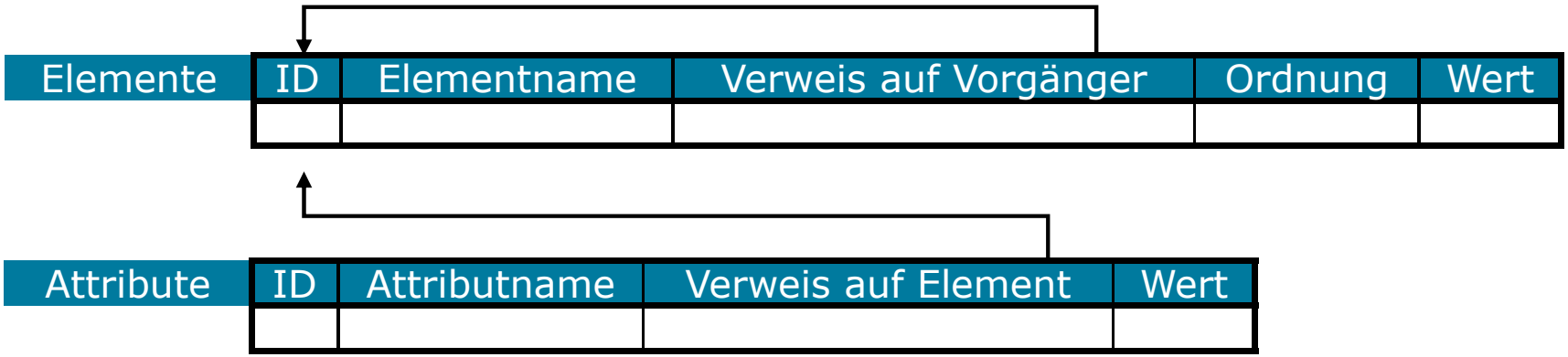
- Textbasierte Verfahren
  - Speicherung der XML-Dokumente als Zeichenkette
- Modellbasierte Verfahren
  - Ausnutzen des Datenmodells von XML zur Speicherung
  - Generische Speicherung der Graphstruktur von XML
  - Speicherung der DOM-Informationen
- Strukturbasierte Verfahren
  - Abbildung auf relationale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

Überblick

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

Modellbasierte Speicherung

- 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


```
<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>
```

Elemente

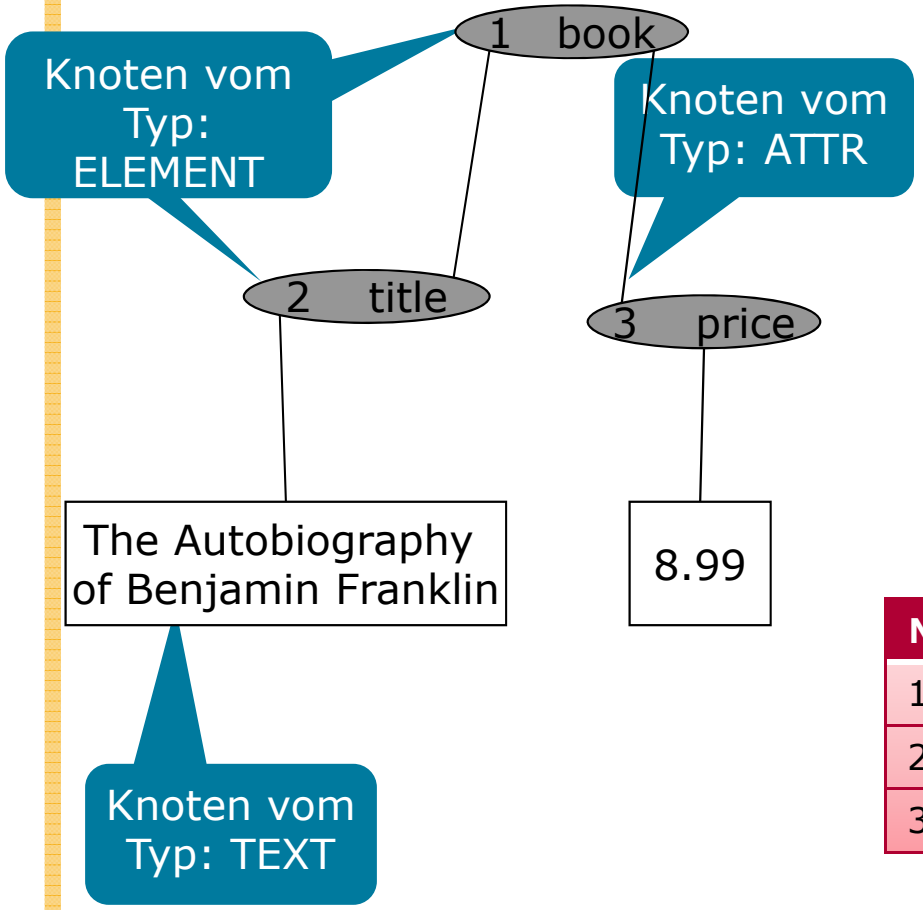
DocID	Elementname	ID	Vorgaenger	Ordnung	Wert
H00001	Hotel	101	⊥	1	⊥
H00001	Hotelname	102	101	1	Hotel Huebner
H00001	Kategorie	103	101	2	4
H00001	Adresse	104	101	3	⊥
H00001	PLZ	105	104	1	18119
H00001	Ort	106	104	2	Warnemuende
H00001	Strasse	107	104	3	Seestrasse
H00001	Nummer	108	104	4	12
H00001	Telefon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

Anfragen

- 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	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

Modellbasierte Speicherung: Speicherung des DOM



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

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

Elementinhalte

NodeID	Content
4	"The Auto..."

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

Überblick

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

Abbildung von XML auf relationale Datenbanken


```
<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


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

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

Hotel_ Kategorie	Hotel_ Hotelname	Pension_ Name	Pension_ Zimmer	Campi ng_St erne	Camping_N ame
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


```
<!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


```
<!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>
```

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>
```

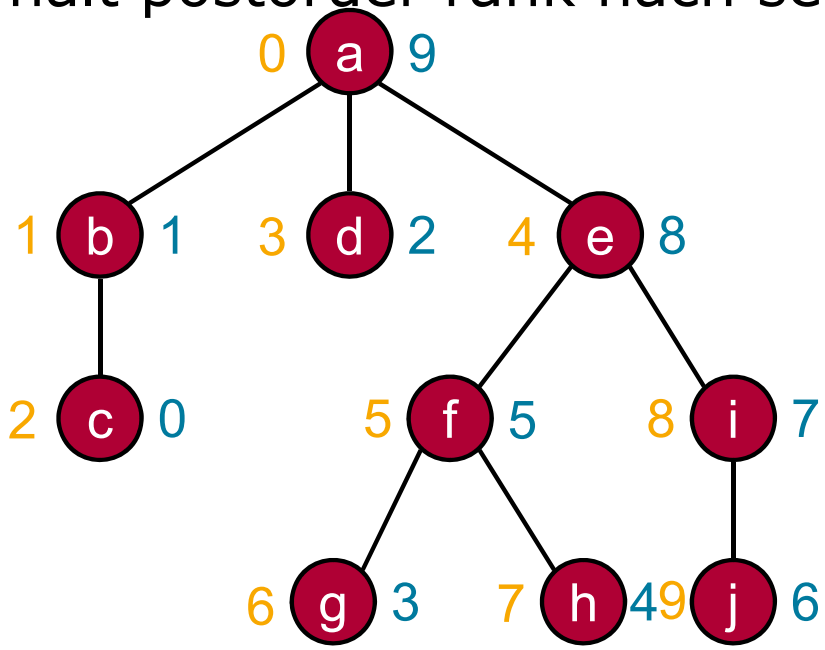
XML-Typ bietet native Methoden zum Zugriff mit XML-Anfragen oder DOM Methoden auf die Daten.

Überblick

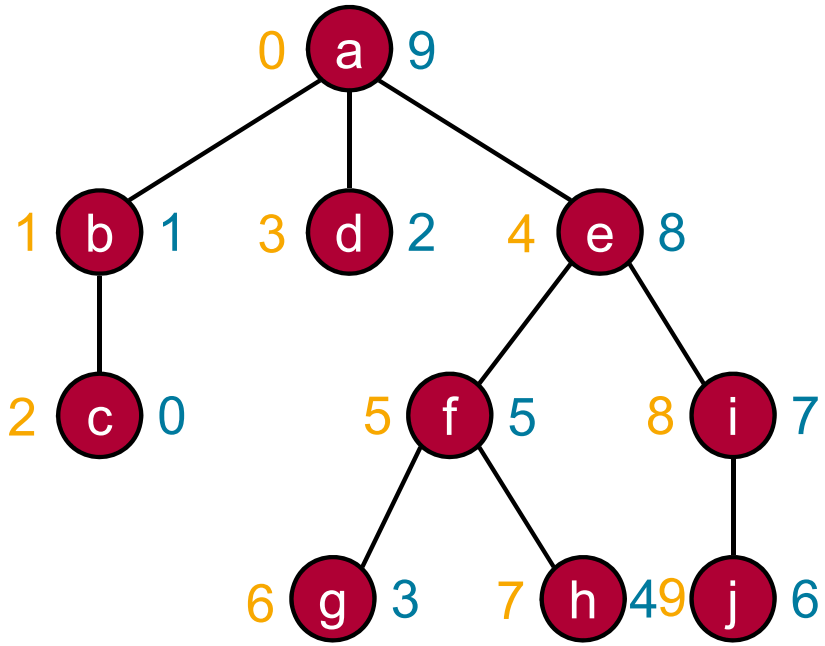
- Motivation & Syntax
- XML Programmierung
- Schemata
- Anfragesprachen
- Speicherung von XML
  - SQL/XML
  - Modell-basierte Verfahren
  - Struktur-basierte Verfahren
  - Indizierung

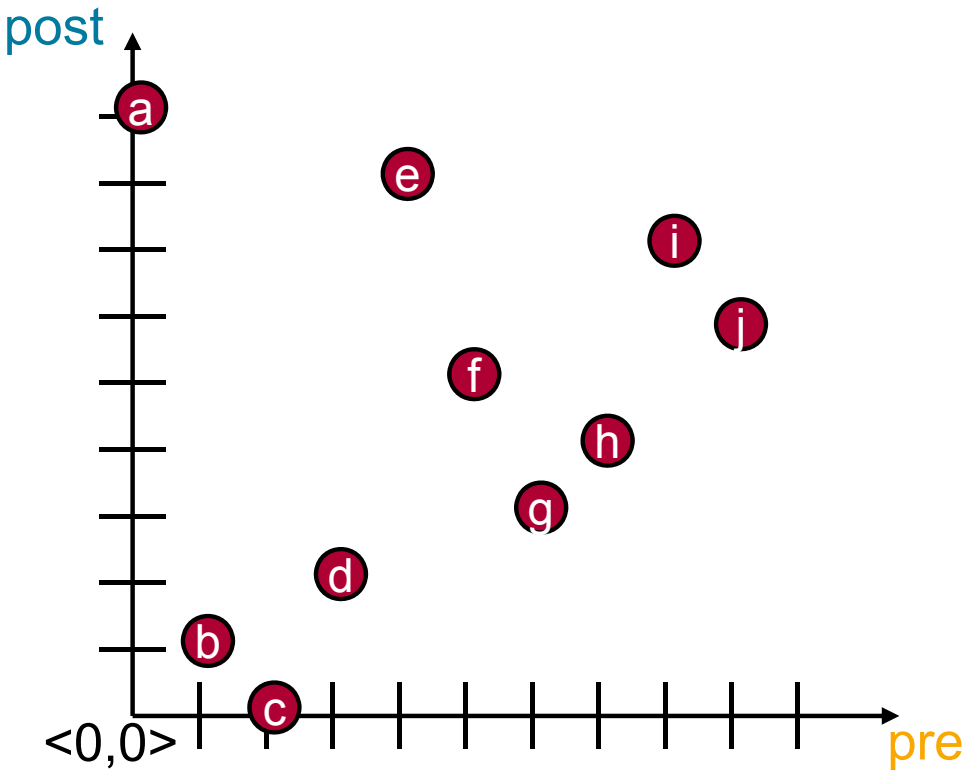
Pre- und Postorder in XML Bäumen

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

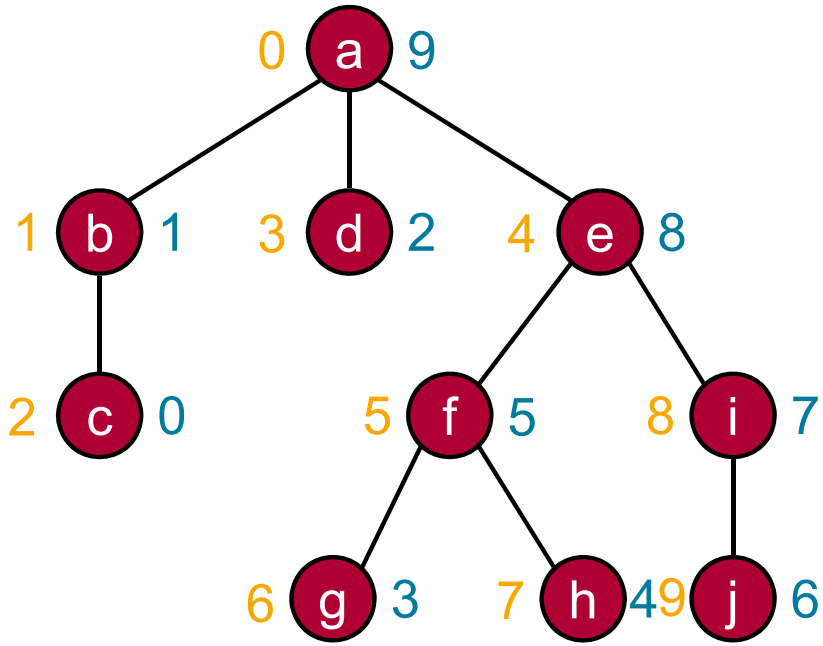


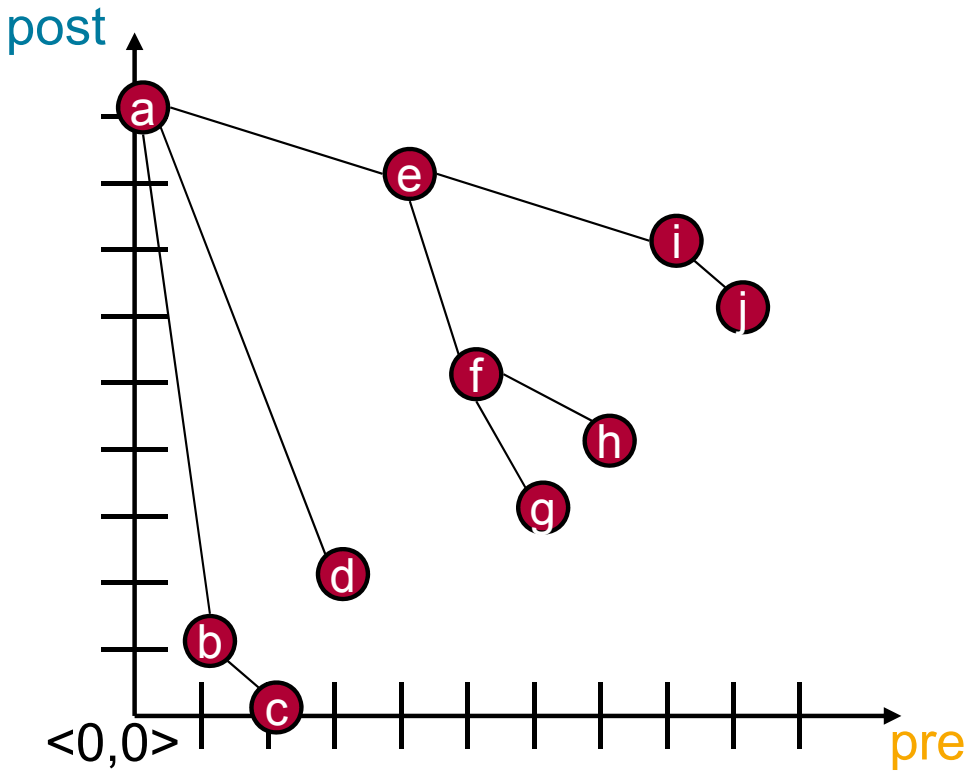
Pre- post-order im Koordinatensystem





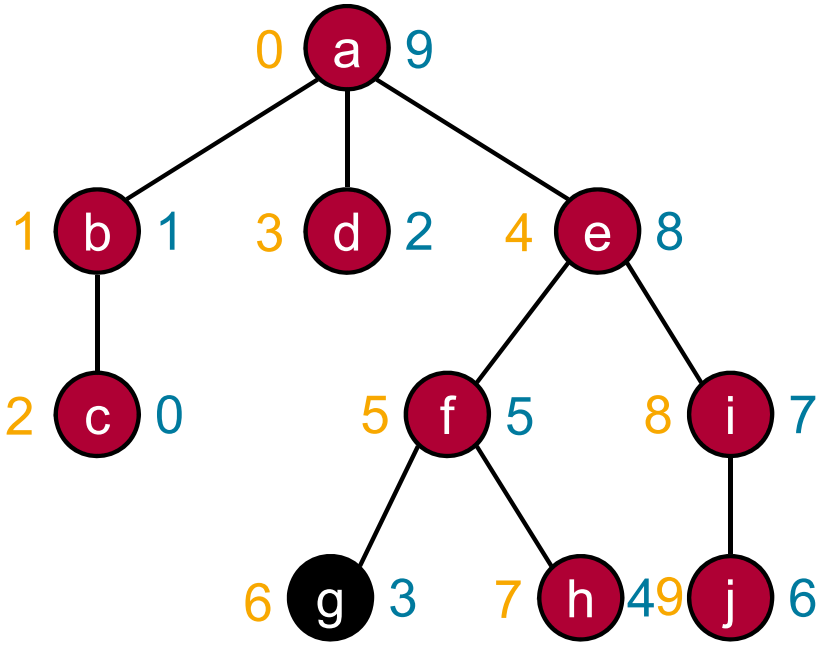
Pre- post-order im Koordinatensystem

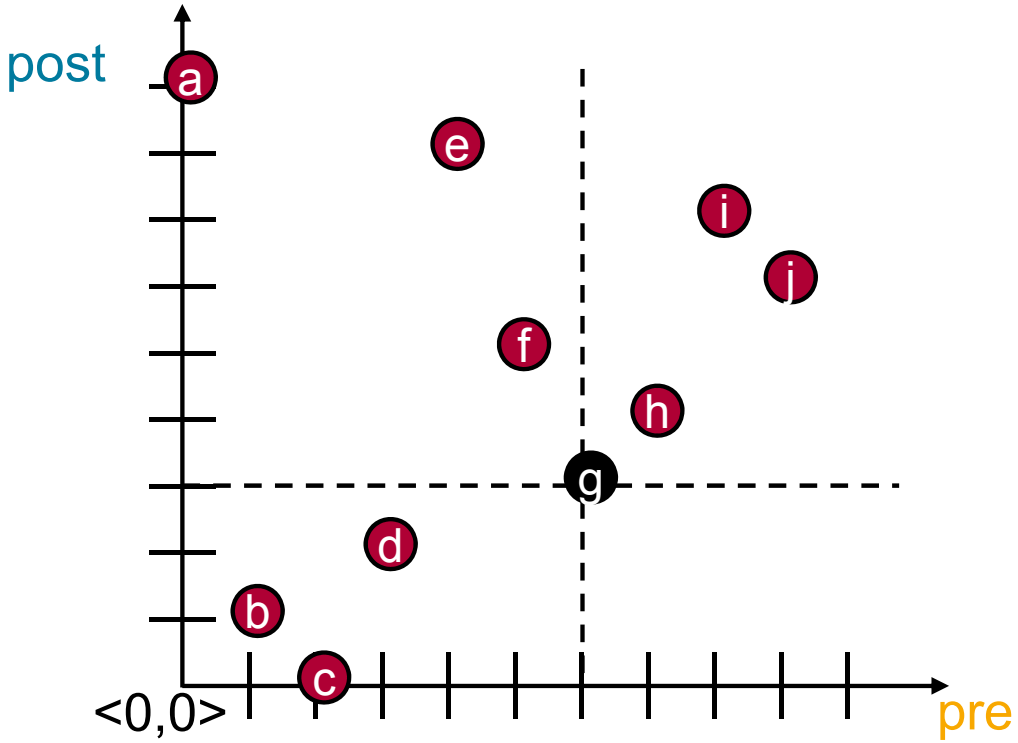




Pre- post-order im Koordinatensystem

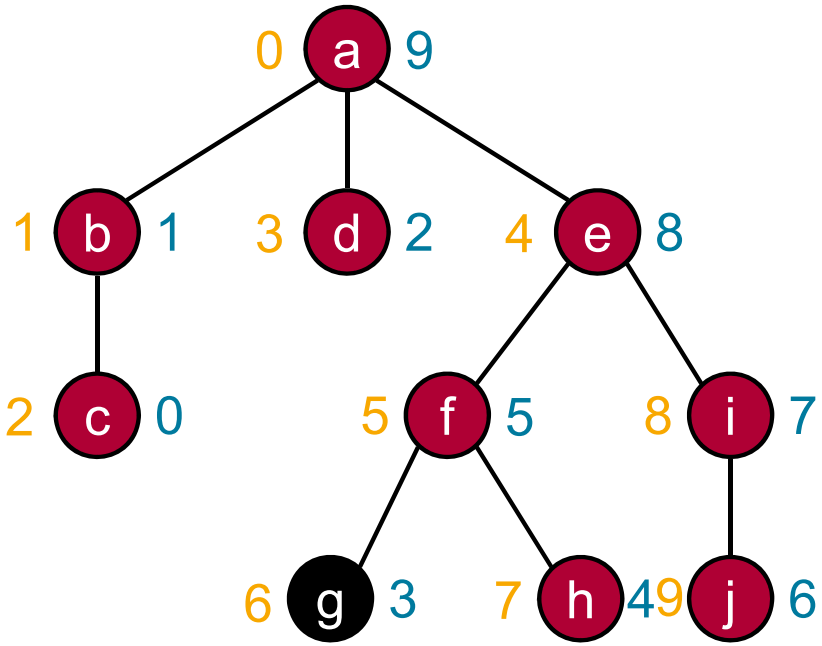
- Jeder Knoten (Kontextknoten) teilt die Ebene in vier Partitionen.

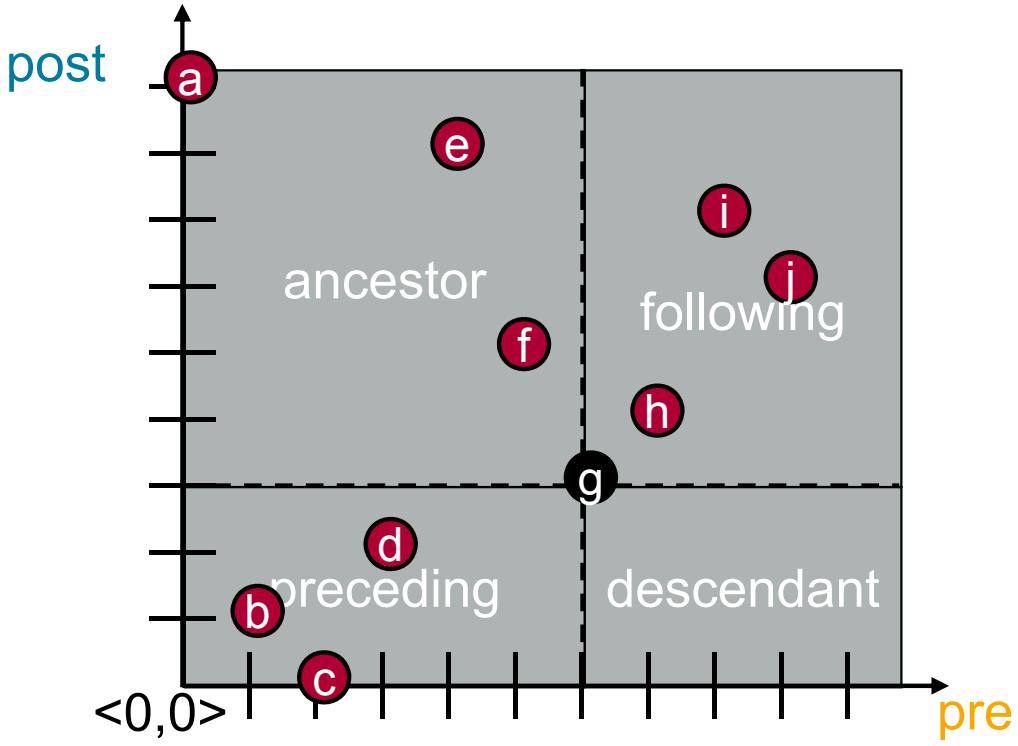




Pre- post-order im Koordinatensystem

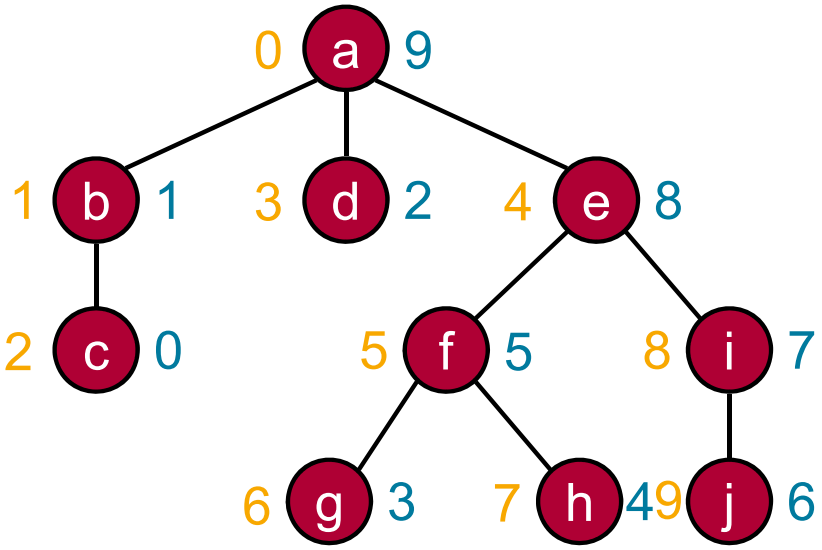
- Verfolgung eines Achs-Schrittes in einer XPath Anfrage entspricht einer rechteckigen Region der Ebene.





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

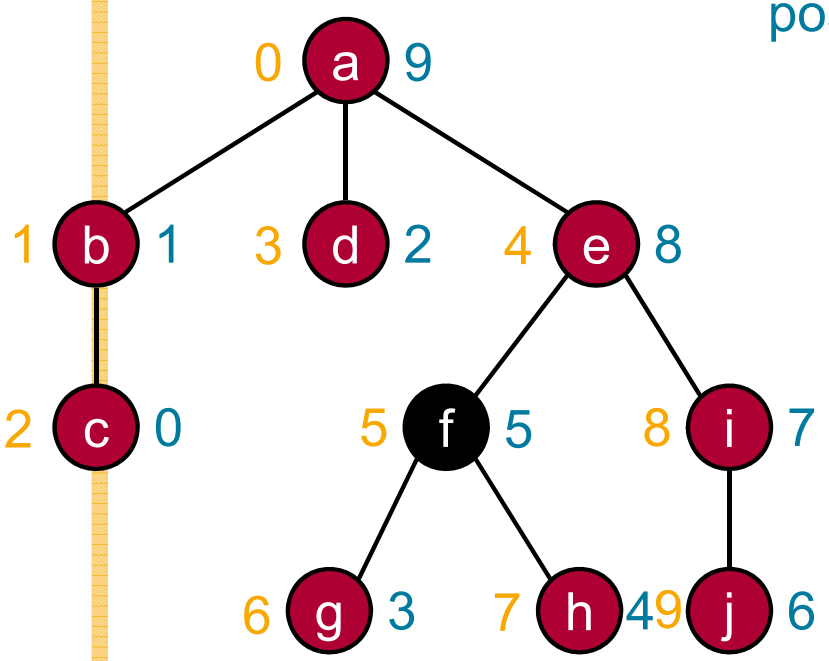
pre	tag
0	a
1	b
4	e
5	f
8	i

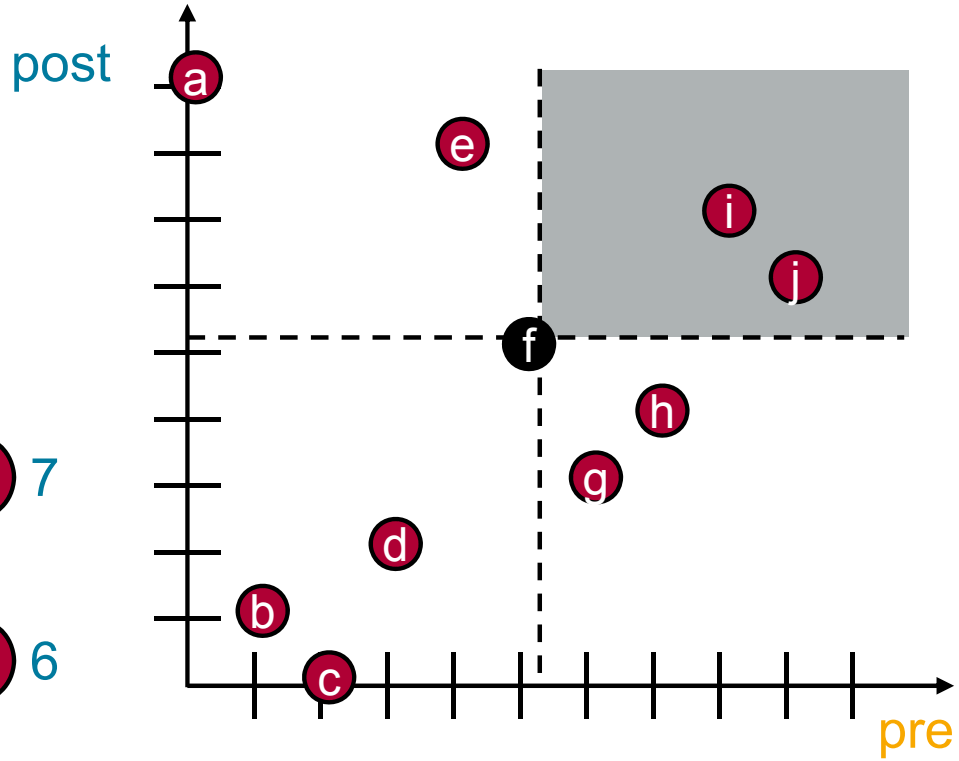
pre	text
2	„c“
7	„h“
9	„j“

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

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
```





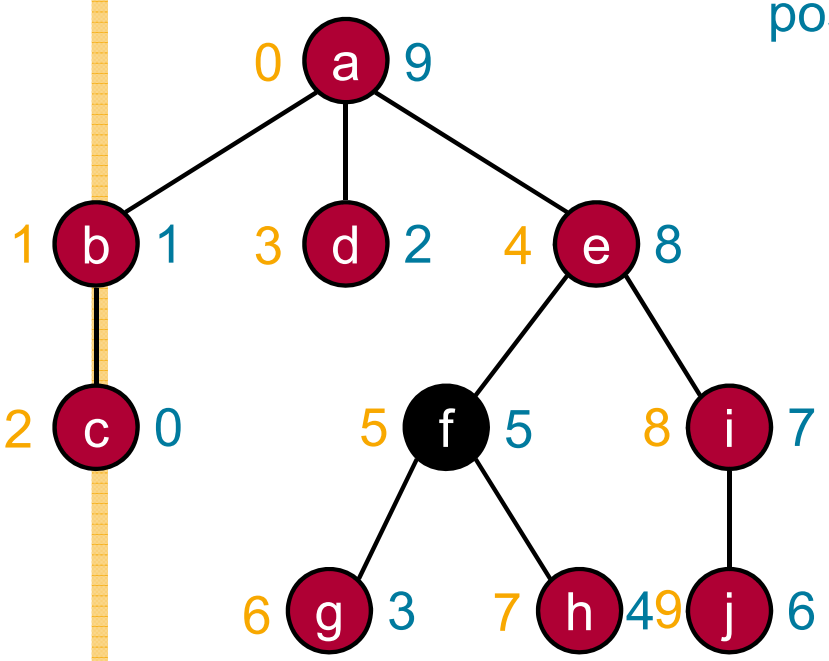
prepost

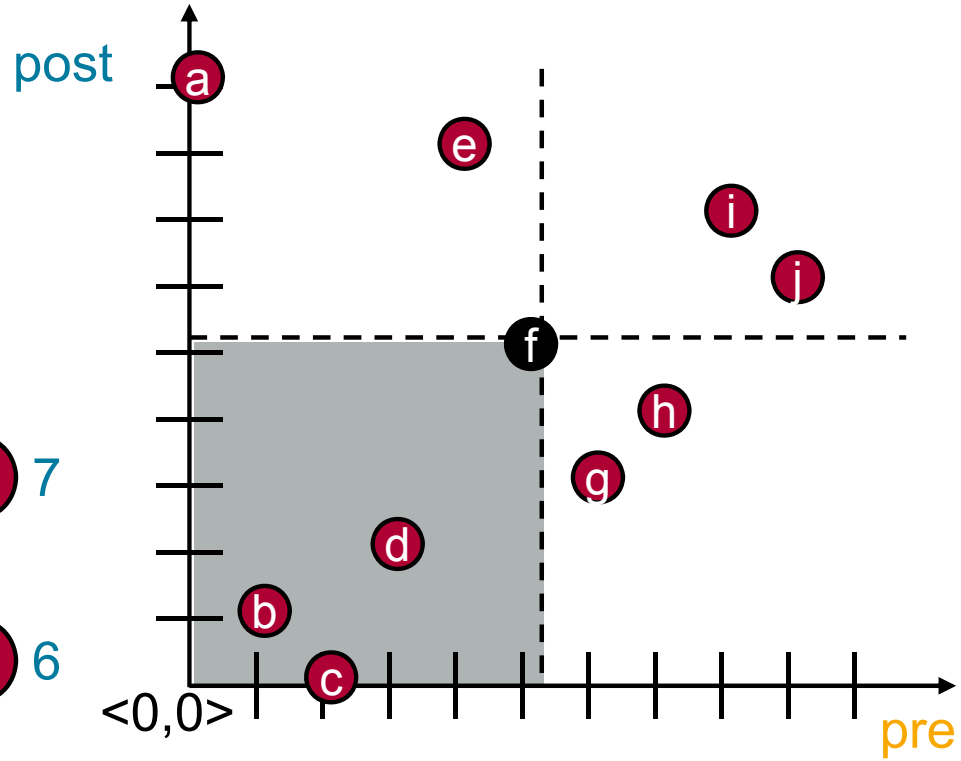
pre	post
0	9
1	1
2	0
3	2
4	8
5	5
6	3
7	4
8	7
9	6

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
```





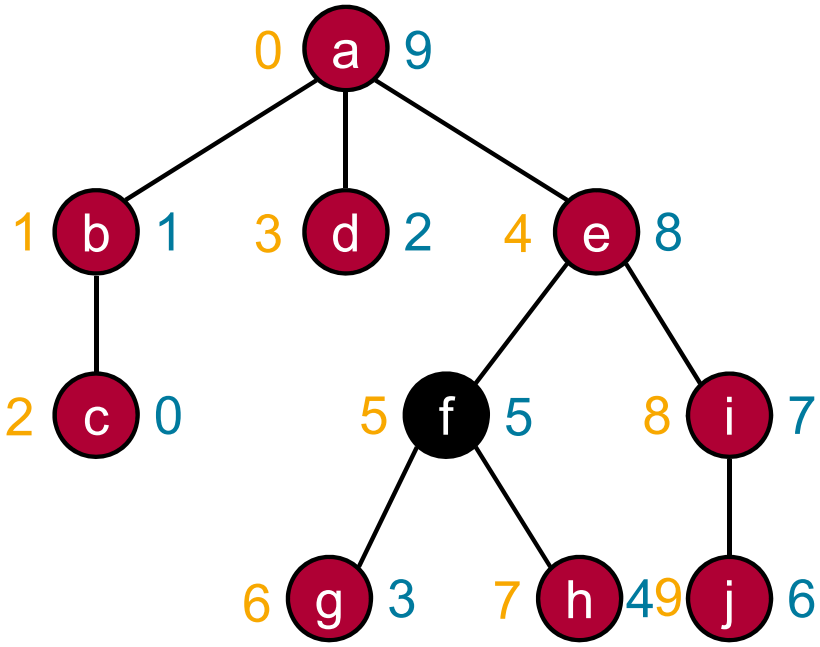
prepost

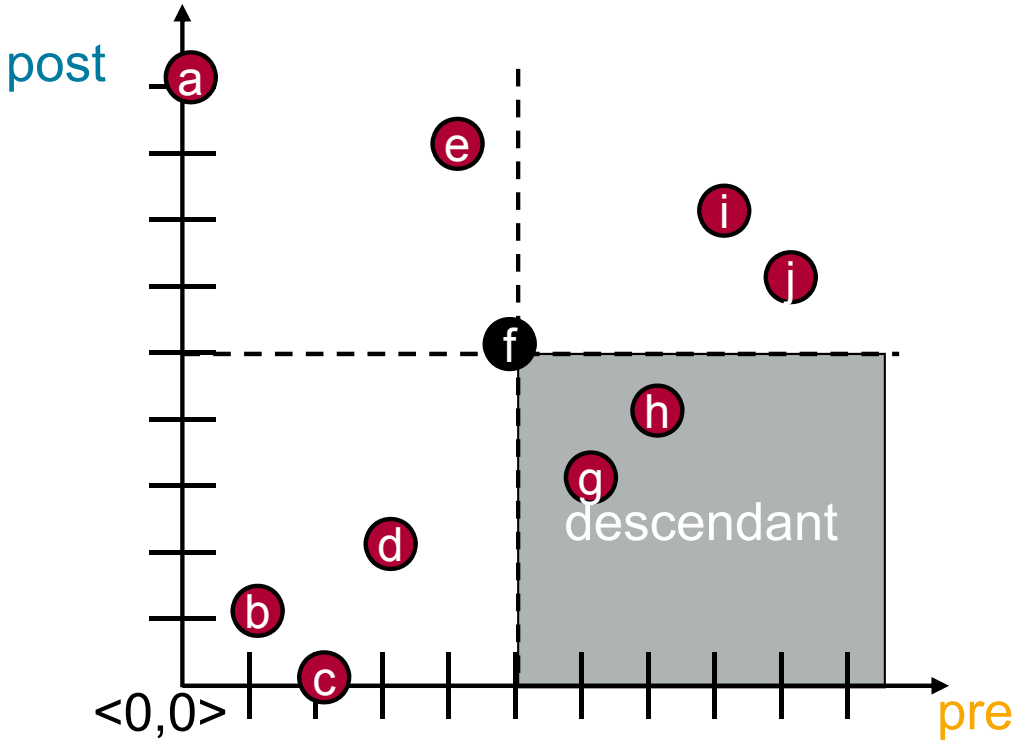
pre	post
0	9
1	1
2	0
3	2
4	8
5	5
6	3
7	4
8	7
9	6

Baum-Bewusstsein

Descendant-Bewusstsein

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

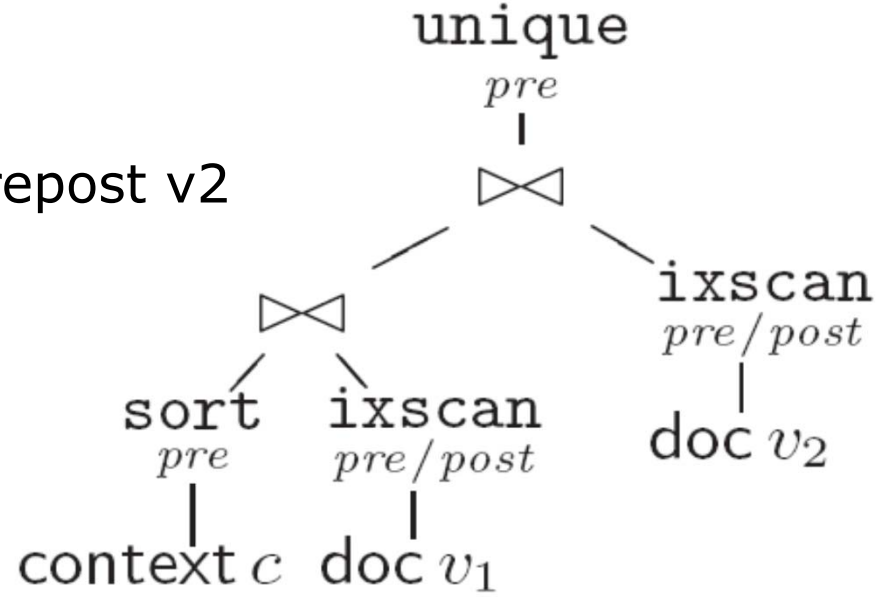




Index-Scans zur Anfragebearbeitung

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

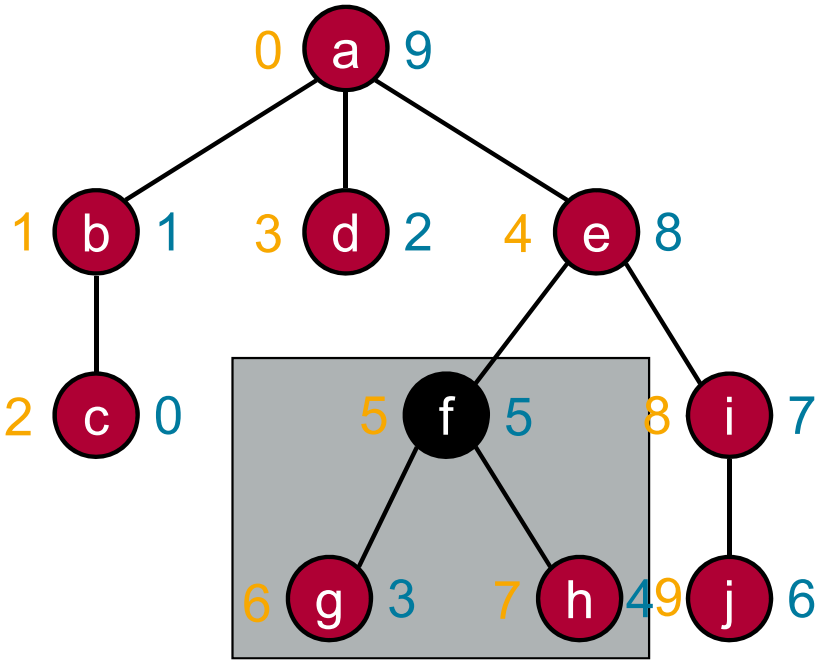


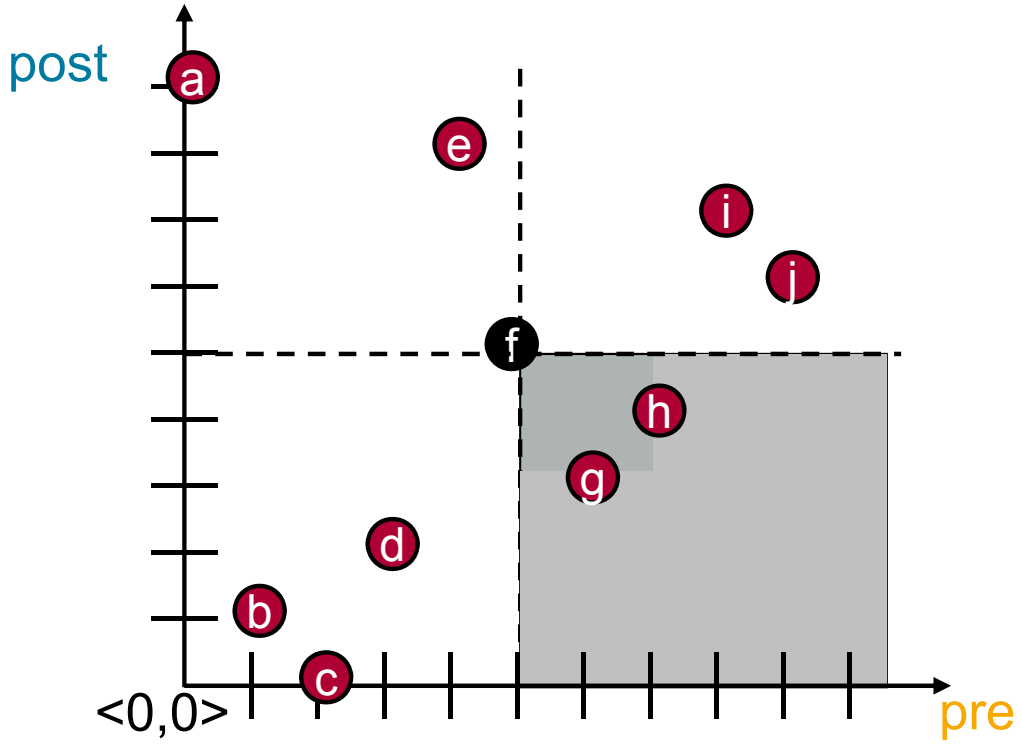
Ausführung im DBMS

- Effizienz gut
- Aber jeweils gesamter Index wird gescannt.

descendant-pruning

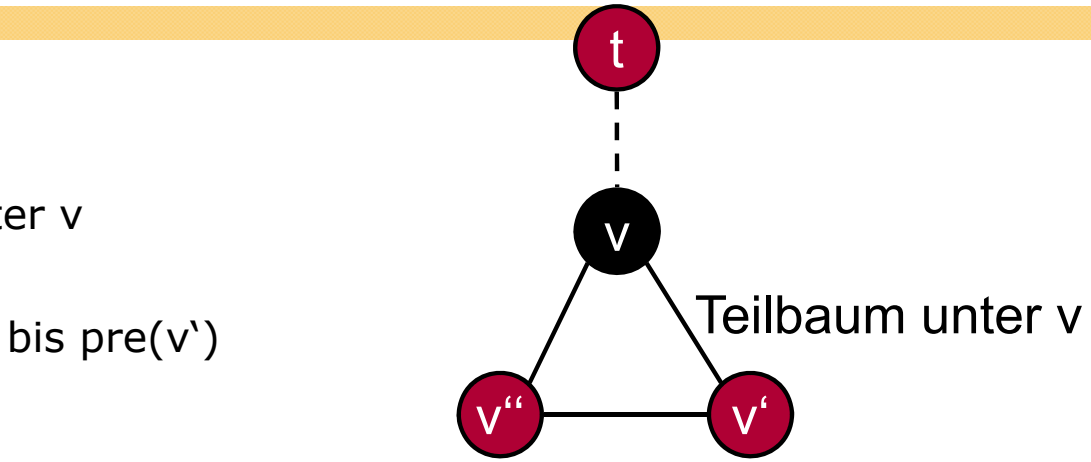
- ... WHERE v*.pre > f.pre AND v*.post < f.post
- Aber:
  - Maximaler pre-Wert im Teilbaum unter f?
    - ⇒ Ab pre > 7 ist der Scan unnötig.
  - Minimaler post-Wert im Teilbaum unter f?
    - ⇒ Ab post < 3 ist der Scan unnötig.
- Problem: Wie nutze ich dieses Baum-Wissen aus?

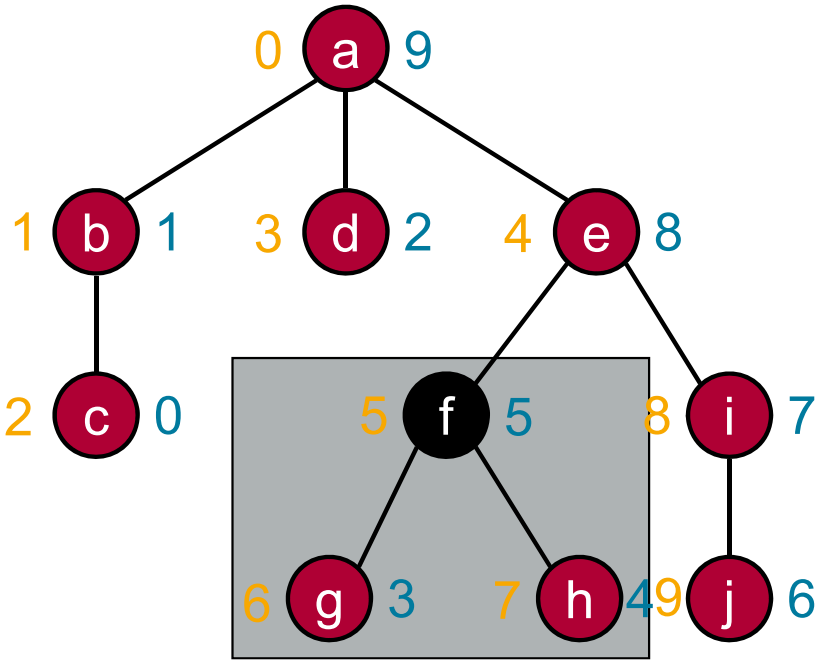


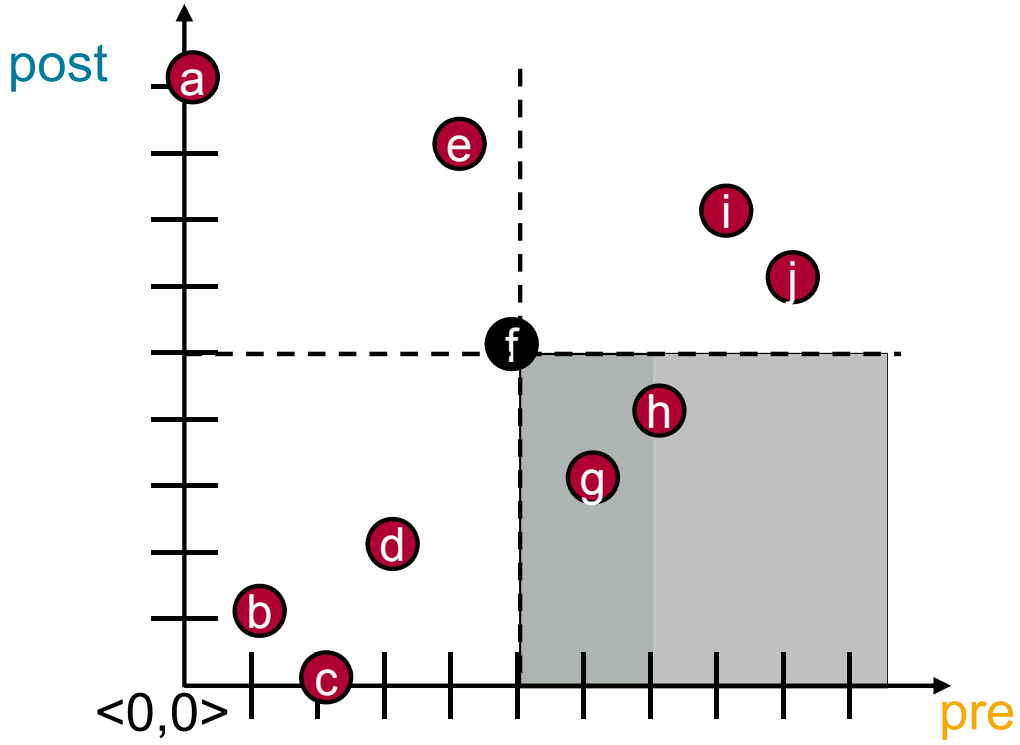


Pruning in der pre-Achse

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

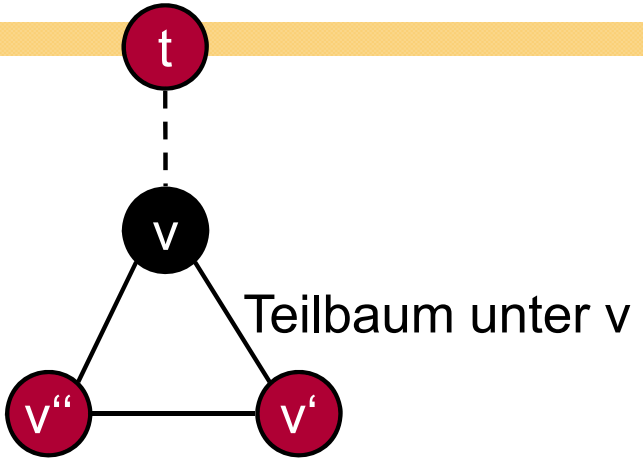


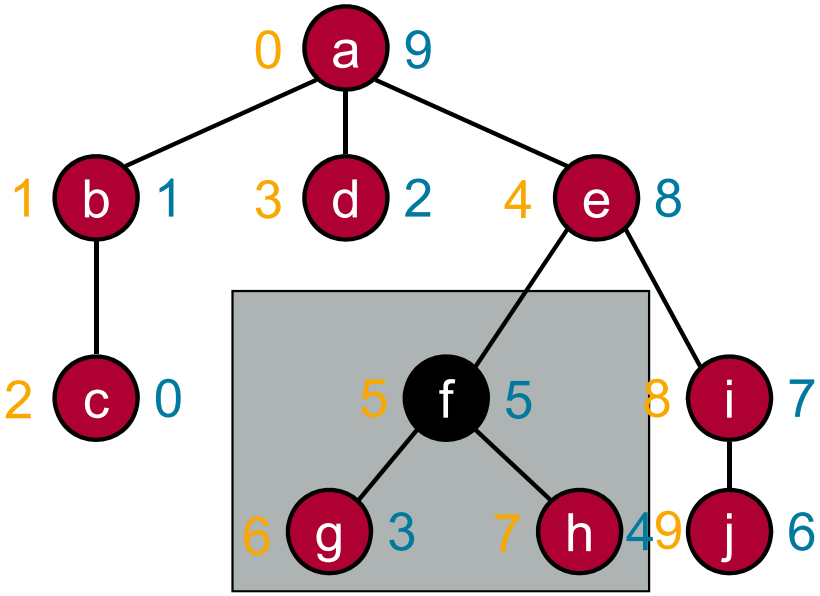


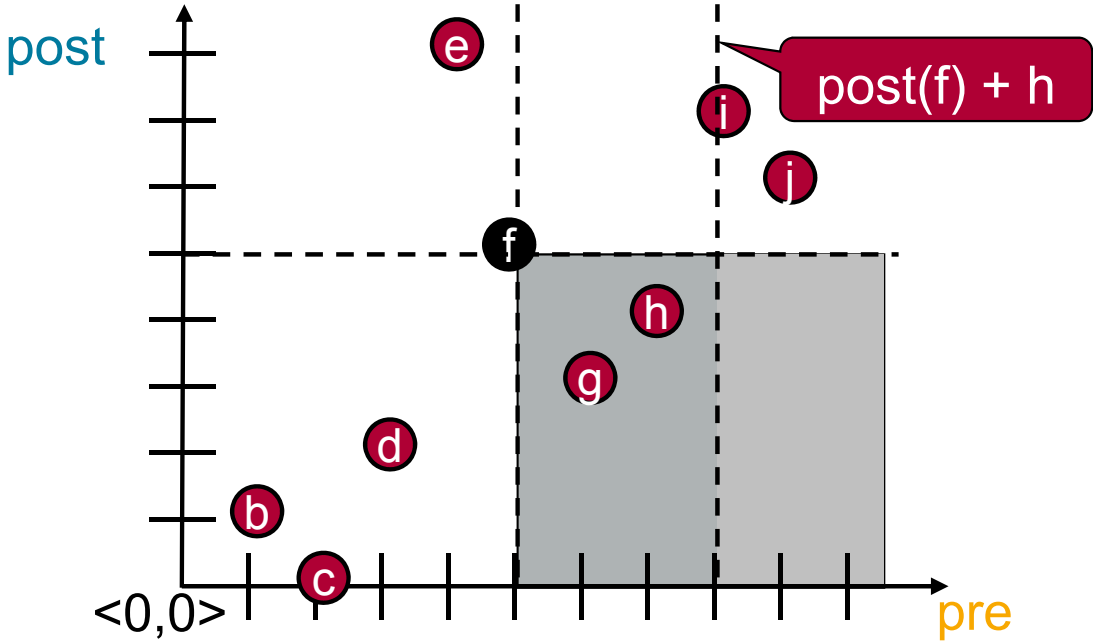


Berechnung pre(v‘)

- pre(v‘) = pre(v) + |v/descendant|
  - Bsp: pre(h) = pre(f) + 2 = 7
- |v/descendant| = post(v) – pre(v) + level(v)
  - = 5 – 5 + 2 = 2
- level(v) ≤ h = Höhe des Gesamtbaums
  - level(f) = 2 ≤ 3
- Zusammen: pre(v‘) ≤ pre(v) + post(v) – pre(v) + h = post(v) + h

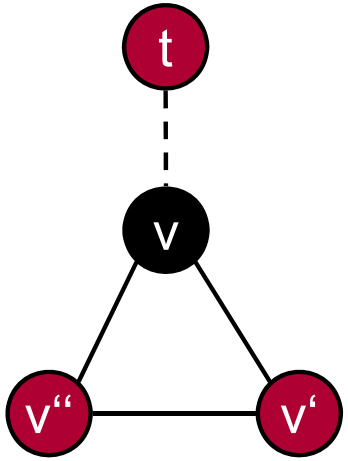


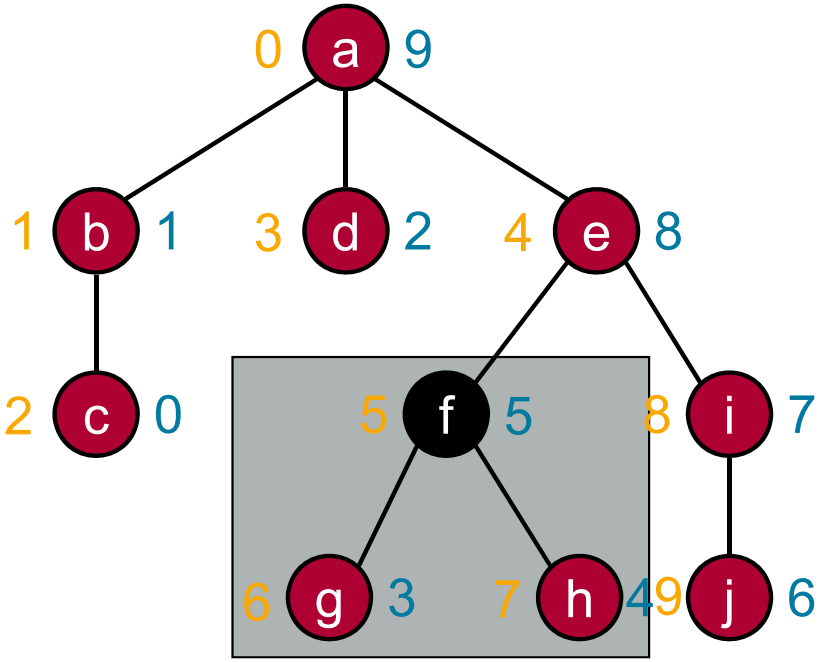


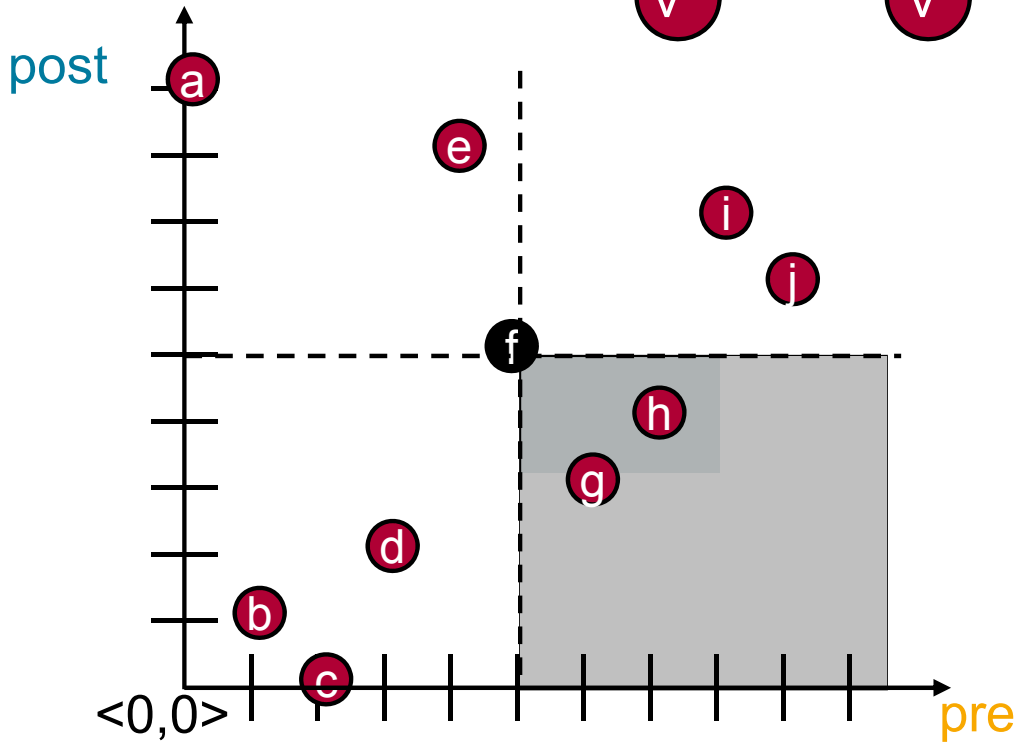


Pruning in der post-Achse

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

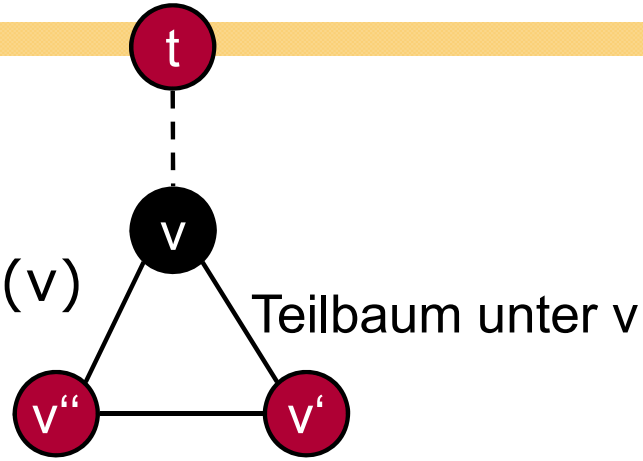


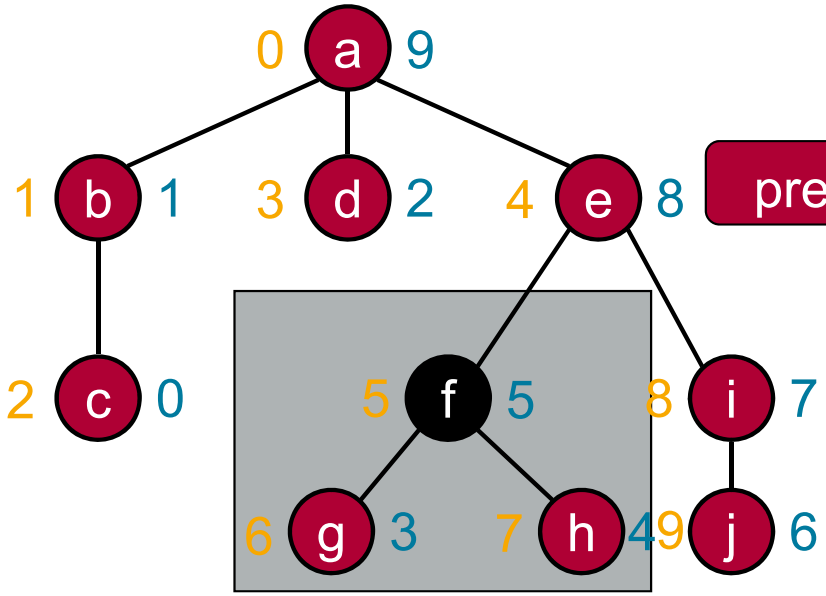


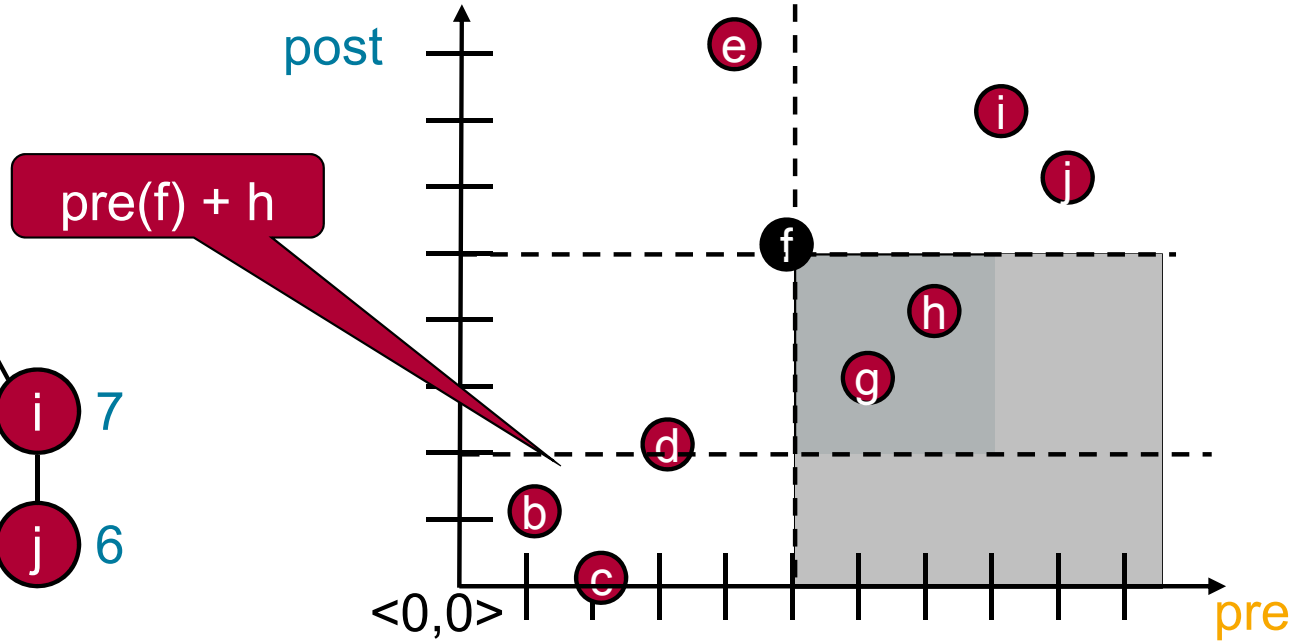


Berechnung post(v‟)

- post(v‟) = post(v) - |v/descendant|
  - Bsp: post(g) = post(f) - 2 = 3
- |v/descendant| = post(v) – pre(v) + level(v)
  - = 5 – 5 + 2 = 2
- level(v) ≤ h = Höhe des Gesamtbaums
  - level(f) = 2 ≤ 3
- Zusammen: post(v‟) ≥ post(v) – (post(v) – pre(v) + h) = pre(v) - h





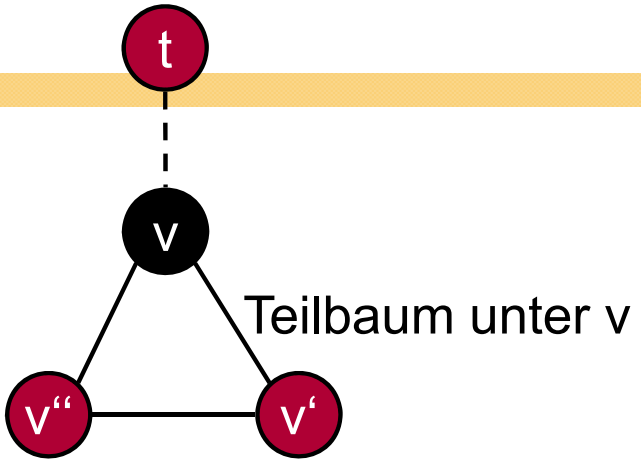


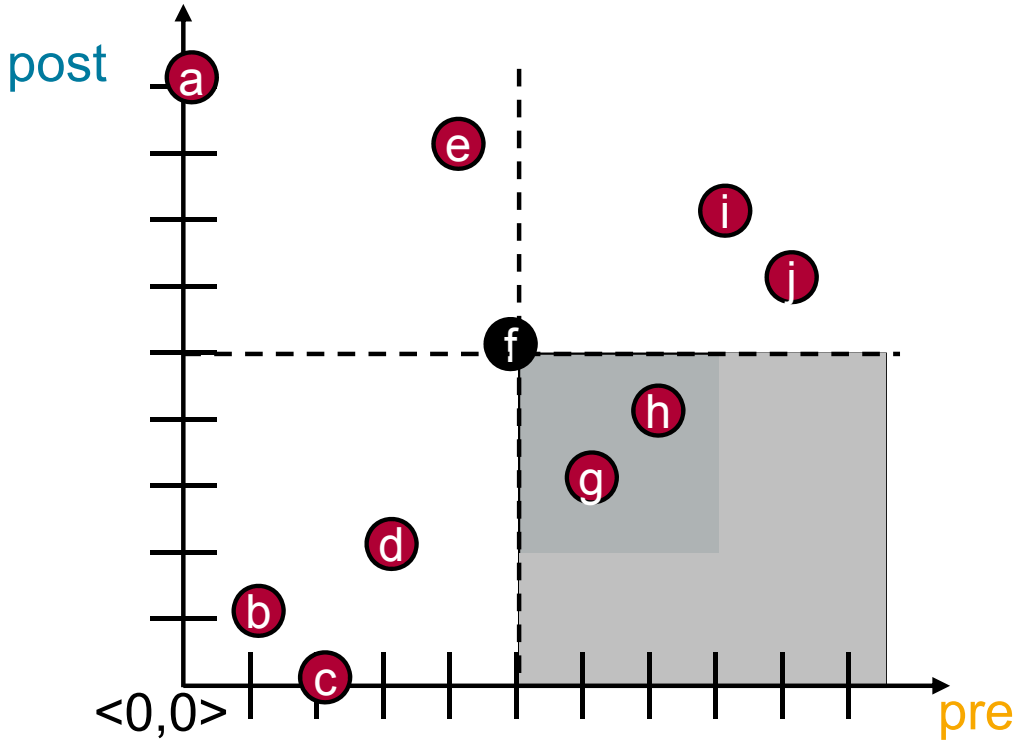
Descendant-pruning

- Insgesamt
  - Vorher

    ...
    WHERE v*.pre > f.pre
    AND v*.post < f.post
  - Nachher

    ...
    WHERE v*.pre > f.pre
    AND v*.pre ≤ f.post +h
    AND v*.post ≥ f.pre +h
    AND v*.post < f.post
- Größe von h
  - Typischerweise < 10
  - Verglichen mit Millionen Knoten





Pruning allgemein

- 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…