# Pflichtmodul Informationssysteme (SS 2014)

Prof. Dr. Jens Teubner

Leitung der Übungen: Marcel Preuß, Sebastian Breß, Martin Schwitalla, Karolina Hilkens

# Lösungsskizze zu Übungsblatt Nr. 13

Ausgabe: 02.07.2014 Abgabe: 09.07.2014

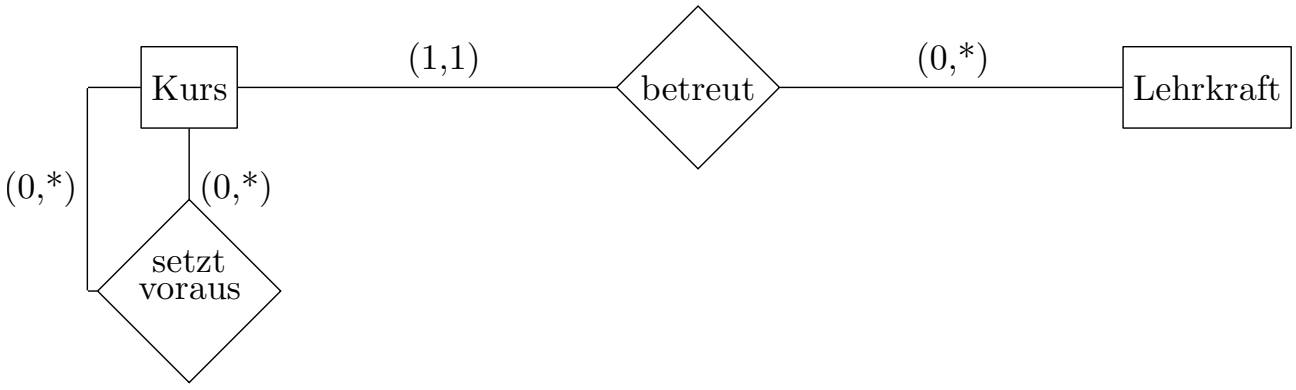
Dieses Übungsblatt ist eine Sammlung von Aufgaben, wie sie auch in einer Klausur gestellt werden könnten (sie wurden teilweise aus alten Klausuren entnommen).

**Aufgabe 1 (ER-Modellierung)**

In einem Schulungszentrum werden verschiedene Kurse angeboten, wobei ausgewählte Kurse den Inhalt anderer Kurse voraussetzen. Je Kurs gibt es genau eine Lehrkraft, die auch verschiedene Kurse betreuen kann.

1. Erstellen Sie für diesen Diskursbereich ein ER-Diagramm. Verwenden Sie für die Funktionalität der Beziehungstypen die Min-/Max-Notation.
2. Überführen Sie das erhaltene ER-Diagramm in Tabellen des Relationenmodells und geben Sie Primär- und Fremdschlüsselbedingungen an.

1. ER-Diagramm:




2. Da keine Attribute vorgegeben sind, fügen wir eigenmächtig welche hinzu (damit die Tabellen interessant werden — in der Klausur tritt so ein Fall nicht auf). Ein Kurs wird durch seine Nummer identifiziert und hat eine Bezeichnung. Eine Lehrkraft wird durch ihre Personalnummer identifiziert und hat einen Namen. Die Relation "setzt voraus" wird über das Paar identifiziert, welches sich aus den Schlüsselattributen der beiden teilnehmenden Entitäten zusammensetzt. Die Relation "betreut" wird durch die Nummer des Kurses identifiziert (dies reicht aufgrund der Kardinalitätsbeschränkung):

In den Tabellen wird folgende Notation verwendet: Die unterstrichenen Attribute bilden jeweils (zusammengenommen im Falle von SetztVoraus) einen Schlüssel für die entsprechende Relation (Tabelle):

(a) Entitäten:

| Kurse | |
|---|---|
| Kursnummer | Bezeichnung |
| ⋮ | ⋮ |

| Lehrkräfte | |
|---|---|
| Personalnummer | Name |
| ⋮ | ⋮ |

(b) Relationen:

| SetztVoraus | |
|---|---|
| Kursnummer | Kursnummer2 |
| ⋮ | ⋮ |

| Betreut | |
|---|---|
| Kursnummer | Personalnummer |
| ⋮ | ⋮ |

Hierbei sind Kursnummer und Kursnummer2 Fremdschlüssel, die auf das entsprechende Attribut in Kurse verweisen. Personalnummer ist ein Fremdschlüssel, der auf das entsrpechende Attribut in Lehrkräfte verweist.

**Aufgabe 2 (Tabellen erzeugen)**

Ein Sportverein will eine Datenbank mit folgendem Schema aufbauen:

*sch(Mitglieder)=(MitglNr, Name, Alter, Abteilung)*
*sch(Beiträge)=(Alter, Betrag)*
*sch(Zahlungen)=(Datum, MitglNr, Betrag)*

In *Beiträge* werden die Mitgliedsbeiträge festgehalten, die vom Alter des Mitglieds abhängig sind. Manche Mitglieder bezahlen ihre Beiträge in Raten, daher sind in *Zahlungen* alle bislang eingegangenen Beträge vermerkt.

1. Geben Sie die SQL-Anweisungen an, mit denen das obige Relationen-Schema erzeugt wird. Spezifizieren Sie dabei sämtliche Primär- und Fremdschlüsselbeziehungen.
2. Trainern soll die Möglichkeit erstellt werden, von allen Mitgliedern *Name* und *Alter* zu sehen. Geben Sie SQL-Kommandos an zur Erstellung einer View, die genau diese Information enthält.

1. SQL-Anweisungen:

```
CREATE TABLE Mitglieder ( MitglNr INTEGER NOT NULL,
                          NameMitglied CHAR(30),
                          AlterMitglied INTEGER,
                          Abteilung CHAR(30),
                          PRIMARY KEY (MitglNr)
)
```

```
CREATE TABLE Beitraege ( AlterMitglied INTEGER NOT NULL,
                         Betrag INTEGER,
                         PRIMARY KEY (AlterMitglied)
)

CREATE TABLE Zahlungen ( Datum DATE NOT NULL,
                         MitglNr INTEGER NOT NULL,
                         Betrag INTEGER,
                         PRIMARY KEY (Datum, MitglNr),
                         FOREIGN KEY (MitglNr) REFERENCES Mitglieder
)
```

2. View:

```
CREATE VIEW NameUndAlter AS SELECT NameMitglied, AlterMitglied FROM Mitglieder;
```

**Aufgabe 3 (Anfragesprachen)**

Gegeben sei das Relationen-Schema aus Aufgabe 2:

*sch(Mitglieder)=(MitglNr, Name, Alter, Abteilung)*
*sch(Beiträge)=(Alter, Betrag)*
*sch(Zahlungen)=(Datum, MitglNr, Betrag)*

Formulieren Sie die folgende Anfrage in **Tupel-Relationen-Kalkül und SQL**. Verwenden Sie `DISTINCT` genau dann, wenn im Ergebnis tatsächlich Duplikate auftreten können.

1. Geben Sie *Name*, *MitglNr* und *Betrag* der fälligen Beiträge aller Mitglieder aus der Abteilung 'Fußball' aus.

   $\{t|\exists m : m \in Mitglieder \wedge \exists b : b \in Beitraege \wedge m.Alter = b.Alter \wedge$
   $m.Abteilung = `Fussball' \wedge t \leftarrow \langle m.Name, m.MitglNr, b.Betrag\rangle\}$

```
SELECT m.NameMitglied, m.MitglNr, b.Betrag
FROM Mitglieder m, Beitraege b
WHERE m.AlterMitglied = b.AlterMitglied AND m.Abteilung = 'Fussball';
```

Formulieren Sie die folgenden beiden Anfragen in **Relationen-Algebra und SQL**. Verwenden Sie auch hier `DISTINCT` genau dann, wenn im Ergebnis tatsächlich Duplikate auftreten können.

2. Welche Mitglieder haben bislang noch gar nichts bezahlt? Geben Sie *MitglNr*, *Name*, *Alter* und den fälligen *Betrag* aus.

   $\Big(\pi_{MitglNr,Name,Alter}(Mitglieder) - \pi_{MitglNr,Name,Alter}(Mitglieder \bowtie Zahlungen)\Big)$
   $\bowtie$
   $Beitraege$

```
SELECT m.MitglNr, m.NameMitglied, m.AlterMitglied, b.Betrag
FROM Mitglieder m, Beitraege b
WHERE m.AlterMitglied = b.AlterMitglied
AND NOT EXISTS (
  SELECT *
  FROM Zahlungen AS z
  WHERE z.MitglNr = m.MitglNr
);
```

3. Geben Sie von den ältesten Mitgliedern *MitglNr* und *Name* aus.

   Self-Join zwischen Mitglieder, wobei linkes Alter größer ist als rechtes. Damit sind alle, die rechts stehen, nicht die ältesten. Wir ziehen sie daher von der Gesamt-Menge an Mitgliedern ab und es bleiben die ältesten. Der erneute Join mit Mitglieder ergibt die passenden Attribute.

   $$\pi_{MitglNr}(Mitglieder) - \pi_{MitglNr \leftarrow m_2}($$
   $$\sigma_{a_1 > a_2}($$
   $$\pi_{m_1 \leftarrow MitglNr, a_1 \leftarrow Alter}(Mitglieder) \times \pi_{m_2 \leftarrow MitglNr, a_2 \leftarrow Alter}(Mitglieder)$$
   $$)$$
   $$)$$
   $$\bowtie$$
   $$\pi_{MitglNr,Name}(Mitglieder)$$

   **Alternativlösung:** Die Idee ist es, zuerst Mitgliedsnummer und Name aller Mitglieder zu berechnen, die nicht ältestes Mitglied des Vereins sind (ähnlich wie oben). Dann zieht man diese Tupel von der Menge aller Tupel von Mitgliedsnummern und Namen ab. Man kriegt dann:

   $$\pi_{MitglNr,Name}(Mitglieder) - \pi_{MitglNr \leftarrow MN_2,Name}(Exp)$$

   Hierbei gelte:

   $$Exp = \sigma_{a_1 > a_2}(\pi_{a_1 \leftarrow Alter}(Mitglieder) \times \pi_{MN_2 \leftarrow MitglNr, a_2 \leftarrow Alter,Name}(Mitglieder))$$

```
SELECT MitglNr, NameMitglied
FROM Mitglieder m
WHERE NOT EXISTS (
  SELECT *
  FROM Mitglieder aelter
  WHERE aelter.Alter > m.Alter
);
```

Formulieren Sie die folgenden zwei Anfragen nur in **SQL**:

4. Welche *Abteilung* hat die meisten Mitglieder, die bisher noch nichts gezahlt haben?

```
select m.abteilung
from mitglieder m where not exists (
   select * from zahlungen z1 where z1.mitglNr=m.mitglNr)
group by m.abteilung having count(*) >= all(
   select count(*)
   from mitglieder m1 where not exists (
      select * from zahlungen z where z.mitglNr = m1.mitglNr)
   group by m1.abteilung);
```

Idee: Wähle die Abteilungen, wo die Anzahl der Mitglieder, die noch nichts gezahlt haben (das sind die, deren Mitgliedsnummer nicht in der Tabelle Zahlungen auftaucht), größer oder gleich der entsprechenden Anzahl jeder einzelnen Abteilung ist.

Eine übersichtlichere Lösung basierend auf Views:

```
CREATE VIEW N AS
  SELECT M.Abteilung
  FROM Mitglieder M
  WHERE NOT EXISTS (
    SELECT *
    FROM Zahlungen Z
    WHERE M.MitglNr = Z.MitglNr
  );



SELECT Abteilung
FROM N
GROUP BY Abteilung
HAVING COUNT(*) >= ALL (
  SELECT COUNT(*)
  FROM N
  GROUP BY Abteilung
);
```

5. Geben Sie für alle Mitglieder, die noch nicht vollständig bezahlt (unter Einbeziehung aller bereits geleisteten Zahlungen) haben, *Name* und den noch verbleibenden Betrag aus.

```
select m.NameMitglied, b.betrag - g.gesamt as verbleibend
from mitglieder m,
     beitraege b,
     (select mitglNr, sum(betrag) as gesamt
      from zahlungen group by mitglNr) g
where g.mitglNr = m.mitglNr and
      b.betrag > g.gesamt and
      b.alterMitglied=m.alterMitglied;
```

**Aufgabe 4 (Transaktionskontrolle)**

Untersuchen Sie die folgenden Schedules auf Serialisierbarkeit. Geben Sie dazu den vollständigen Abhängigkeitsgraphen an und beschriften Sie die Kanten mit allen vorkommenden Konflikten. Geben Sie, falls möglich, einen äquivalenten seriellen Schedule an.

(a) $\langle r_2(x), r_3(y), w_2(x), r_4(x), w_3(z), r_1(z), w_1(z), r_4(z), w_4(x), r_2(y), w_4(z), w_2(y)\rangle$

(b) $\langle r_3(u), r_2(v), w_3(u), r_2(u), r_1(w), w_2(v), r_1(v), r_3(u), r_3(w), r_2(u), w_3(w), w_2(u)\rangle$

**Lösung:**

1. $\langle r_2(x), r_3(y), w_2(x), r_4(x), w_3(z), r_1(z), w_1(z), r_4(z), w_4(x), r_2(y), w_4(z), w_2(y)\rangle$
   Konfliktrelation:

   $r_2(x) \prec w_4(x)$ Label für Kante 1
   $r_3(y) \prec w_2(y)$ Label für Kante 2
   $w_2(x) \prec r_4(x)$ Label für Kante 1
   $w_2(x) \prec w_4(x)$ Label für Kante 1
   $w_3(z) \prec r_1(z)$ Label für Kante 3
   $w_3(z) \prec w_1(z)$ Label für Kante 3
   $w_3(z) \prec r_4(z)$ Label für Kante 4
   $w_3(z) \prec w_4(z)$ Label für Kante 4
   $r_1(z) \prec w_4(z)$ Label für Kante 5
   $w_1(z) \prec r_4(z)$ Label für Kante 5
   $w_1(z) \prec w_4(z)$ Label für Kante 5

   Konfliktgraph:

   - $T_2 \to T_4$ (Kante 1)
   - $T_3 \to T_2$ (Kante 2)
   - $T_3 \to T_1$ (Kante 3)
   - $T_3 \to T_4$ (Kante 4)
   - $T_1 \to T_4$ (Kante 5)

   Der Graph ist azyklisch. Also gibt es konflikt-äquivalente serielle Ausführung, zB: $\langle T_3, T_1, T_2, T_4\rangle$.

2. $\langle r_3(u), r_2(v), w_3(u), r_2(u), r_1(w), w_2(v), r_1(v), r_3(u), r_3(w), r_2(u), w_3(w), w_2(u)\rangle$
   Konfliktrelation:

   $r_3(u) \prec w_2(u)$ Label für Kante 1
   $w_3(u) \prec r_2(u)$ Label für Kante 1
   $w_3(u) \prec w_2(u)$ Label für Kante 1
   $r_1(w) \prec w_3(w)$ Label für Kante 2
   $w_2(v) \prec r_1(v)$ Label für Kante 3

   Konfliktgraph:

   - $T_3 \to T_2$ (Kante 1)
   - $T_1 \to T_3$ (Kante 2)
   - $T_2 \to T_1$ (Kante 3)

   Der Graph hat einen Zyklus. Also ist der Schedule nicht konflikt-serialisierbar.

**Aufgabe 5 (Schemanormalisierung)**

Gegeben seien das Relationenschema

$$sch(R) = VWXYZ$$

sowie die zugehörige Menge

$$\mathcal{F} = \{Z \to X, V \to W, X \to YV, W \to V\}$$

von funktionalen Abhängigkeiten.

1. Geben sie einen Schlüssel für $sch$(R) an!
2. **Zerlegen Sie $R$ mit Hilfe des BCNF-Zerlegungsalgorithmus aus der Vorlesung, sodass alle resultierenden Tabellen in BCNF vorliegen.**

   ▷ Geben Sie vor jedem Durchlauf der **while**-Schleife sowie nach Ablauf des Algorithmus die Schemata aller erzeugten Tabellen an. Listen Sie dabei auch alle zugehörigen Funktionalen Abhängigkeiten auf.

**Lösung:** Wir haben $sch(R) = VWXYZ$ und $\mathcal{F} = \{Z \to X, V \to W, X \to YV, W \to V\}$. Ein Schlüssel für $R$ ist $Z$, da durch $\{Z \to X\}$ $X$ abgeleitet werden kann, gefolgt von $YV$ durch $\{X \to YV\}$, und $W$ durch $\{V \to W\}$. Da immer $\{Z \to Z\}$ gilt, kann durch Kenntnis von $Z$ jedes Attribut in $sch(R)$ abgeleitet werden, woraus folgt das $Z$ ein Schlüssel ist.

Die drei funktionalen Abhängigkeiten $V \to W, X \to YV$ und $W \to V$ verletzen die BCNF-Eigenschaft. Wir listen alle Möglichkeiten auf, den Nichtdeterminismus aufzulösen (es genügt der Lösungsweg zu einer Menge von Schemata in BCNF!!!):

1. Entlang $V \to W$ zerlegen. Es ergibt sich die Menge:

$$\{(VW, \{V \to W, W \to V\}), (VXZY, \{X \to YV, Z \to X\})\}$$

   Beim zweiten Schema verletzt die Abhängigkeit $X \to YV$ die Bedingung.

   (a) Entlang $X \to YV$ zerlegen. Es ergibt sich die Menge:

$$\{(VW, \{V \to W, W \to V\}), (XYV, X \to YV), (ZX, Z \to X)\}$$

   Alle Schemata sind in BCNF

2. Entlang $W \to V$ zerlegen. Es ergibt sich die Menge:

$$\{(VW, \{V \to W, W \to V\}), (WXZY, \{X \to Y, Z \to X\})\}$$

   Beim zweiten Schema verletzen beide Abhängigkeiten die Bedingung.

   (a) Entlang $X \to Y$ zerlegen. Es ergibt sich die Menge:

$$\{(VW, \{V \to W, W \to V\}), (XY, \{X \to Y\}), (WXZ, \{Z \to X\})\}$$

   Beim letzten Schema verletzt $Z \to X$ die Bedingung.

   i. Entlang $Z \to X$ zerlegen. Es ergibt sich die Menge:

$$\{(VW, \{V \to W, W \to V\}), (XY, \{X \to Y\}), (ZX, \{Z \to X\}), (WZ, \emptyset)\}$$

   Alle Schemata sind in BCNF

   (b) Entlang $Z \to X$ zerlegen. Es ergibt sich die Menge:

$$\{(VW, \{V \to W, W \to V\}), (ZX, \{Z \to X\}), (WZY, \emptyset)\}$$

   Alle Schemata sind in BCNF.

3. Entlang $X \to YV$ zerlegen. Es ergibt sich die Menge:

$$\{(XYV, \{X \to YV\}), (WXZ, \{Z \to X\})\}$$

   Beim zweiten Schema verletzt $Z \to X$ die Bedingung.

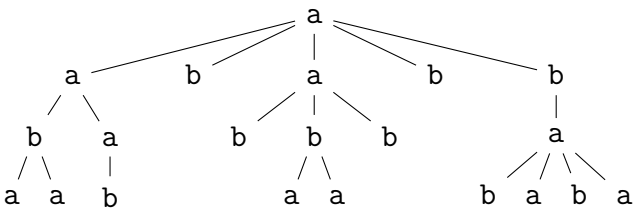
   (a) Entlang $Z \to X$ zerlegen. Es ergibt sich die Menge:

$$\{(XYV, \{X \to YV\}), (ZX, \{Z \to X\}), (WZ, \emptyset)\}$$

   Alle Schemata sind in BCNF.

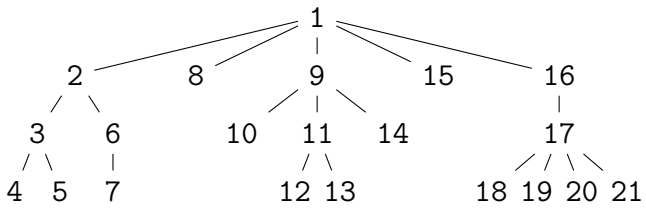
**Aufgabe 6 (XML-Datenbanken / XPath)**

Gegeben sei ein XML-Fragment, dargestellt hier als Baum:




1. Nummerieren Sie alle Knoten in *document order*.

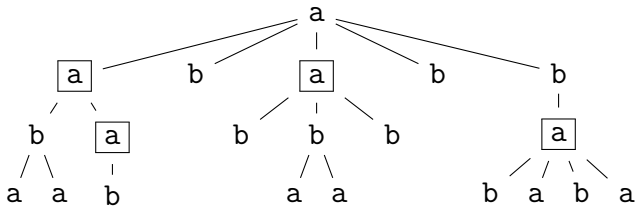
   Die Document Order ergibt sich aus einer Tiefensuche (von links nach rechts) im Baum. Dies entspricht der Reihenfolge der öffnenden XML-Tags. Wir erhalten:




2. Nehmen Sie an, das *context item* '`.`' ist an den Wurzelknoten (mit Namen '`a`') des Fragments gebunden. Welche(r) Knoten wird durch folgende XPath-Ausdrücke zurückgeliefert? Geben sie für jeden XPath Ausdruck die Zwischenschritte in derselben Notation wie auf Folie 311 und 312 an, die sich durch die Anwendung des / Operators ergeben! Geben sie als Ergebnisknoten die Knotennummer bezüglich der *document order* an!

   (a) `./descendant::a[b]`

   Selects all nodes on a level below the root-node (not the root node itself) with label `a` that have a child with label `b`:




   Same as `./descendant::a[child::b]` (diese abkürzende Schreibweise findet sich auf Folie 327: Wenn der '*axis*`::`'-Teil fehlt, wird er standardmäßig durch '`child::`' ersetzt).

   Die Zwischenschritte sind wie folgt:

   i. Die descendant Achse liefert als Zwischenergebnis die folgenden Knoten zurück:

   $$\texttt{./descendant::*} = \{2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21\}$$

   ii. Davon qualifizieren sich für den Nodetest a die folgenden Knoten:

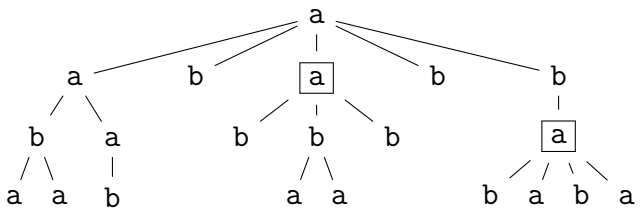
   $$\texttt{./descendant::a} = (2, 4, 5, 6, 9, 12, 13, 17, 19, 21)$$

iii. Davon suchen wir die Knoten, die ein b als Kind haben:

`./descendant::a[child::b]` $= (2, 6, 9, 17)$

(b) `./descendant::a[child::b[2]]`

Selects all nodes on a level below the root-node (not the root node itself) with label `a` having a second child with label `b`.




Die Zwischenschritte sind wie folgt:

i. Die descendant Achse liefert als Zwischenergebnis die folgenden Knoten zurück:

`./descendant::*` $= (2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21)$

ii. Davon qualifizieren sich für den Nodetest a die folgenden Knoten:

`./descendant::a` $= (2, 4, 5, 6, 9, 12, 13, 17, 19, 21)$

iii. Davon suchen wir die Knoten, die ein b als Kind haben:

`./descendant::a[child::b]` $= (2, 6, 9, 17)$

iv. Darauf wenden wir das Prädikat 2 an, die zwei b als Kindknoten haben:

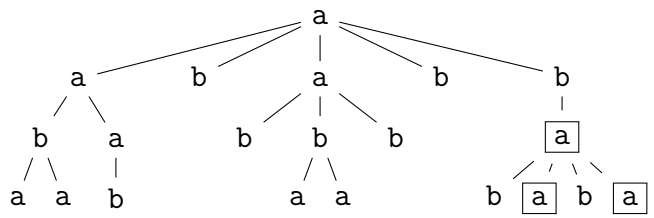
`./descendant::a[child::b[2]]` $= (9, 17)$

(c) `./b//a[ancestor::a]`

Corresponds to:

‘`./child::b/descendant-or-self::node()/child::a[ancestor::a]`’

(Note that `//` expands to `/descendant-or-self::node()/`, cf. slide 327). This expression selects all nodes labeled `a` in the rightmost subtree:




Die Zwischenschritte sind wie folgt:

i. Die child Achse liefert als Zwischenergebnis die folgenden Knoten zurück:

`./child` $= (2, 8, 9, 15, 16)$

ii. Davon qualifizieren sich für den Nodetest b die folgenden Knoten:

`./child::b` = (8, 15, 16)

iii. Davon holen wir alle Folgeknoten, den aktuellen Knoten jeweils mit eingeschlossen:

`./child::b/descendant-or-self::node()` = (8, 15, 16, 17, 18, 19, 20, 21)

iv. Davon suchen wir die Knoten, die als Kind ein a haben:

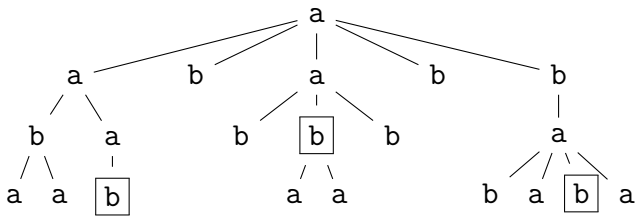
`./child::b/descendant-or-self::node()/child::a` = (17, 19, 21)

v. Darauf wenden wir das Prädikat ancestor::a an, welches testet, ob ein a oberhalb der Hierarchie vorkommt:

`./child::b/descendant-or-self::node()/child::a[ancestor::a]` = (17, 19, 21)

(d) `./descendant-or-self::a/descendant::b[2]`

Selects all the second descendant with label `b` of all nodes with label `a` (this time the root is included — because of the use of `descendant-or-self::` instead of `descendant::`):




Die Zwischenschritte sind wie folgt:

i. Die descendant-or-self Achse liefert als Zwischenergebnis die folgenden Knoten zurück:

`./descendant-or-self::*` = (1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21)

ii. Davon qualifizieren sich für den Nodetest a die folgenden Knoten:

`./descendant-or-self::a` = (1, 2, 4, 5, 6, 9, 12, 13, 17, 19, 21)

iii. Die descendant Achse liefert als Zwischenergebnis die folgenden Knoten zurück:

`./descendant-or-self::a/descendant::*` = (2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21)

iv. Der Ausdruck descendant::b liefert dann nur die Folgeknoten, die ein b haben:

`./descendant-or-self::a/descendant::b` = (3, 7, 8, 10, 11, 14, 15, 16, 18, 20)

v. Der finale XPath Ausdruck liefert nun die Knoten zurück, die die zweiten b Kindknoten der vorherigen Knoten sind (Dabei entfallen die Knoten, die keine zwei Kindknoten haben, zu beachten ist auch, dass das transitiv gilt: z.B. ist von Knoten zwei der zweite b Kindknoten 7, und Kind 7 ist der "Enkel" von 2):

`./descendant-or-self::a/descendant::b[2]` = (7, 11, 20)