Klausur "Semistrukturierte Daten und XML"
Sommersemester 2011
Prof. Dr. Wolfgang May
14. Juli 2011, 14-16 Uhr
Bearbeitungszeit: 90 Minuten

Vorname:

Nachname:

Matrikelnummer:

Bei der Klausur sind **keine Hilfsmittel** (Skripten, Taschenrechner etc.) erlaubt. Handies müssen ausgeschaltet sein. Papier wird gestellt. Benutzen Sie nur die **ausgeteilten**, zusammengehefteten **Blätter** für Ihre Antworten. Schreiben Sie mit blauem/schwarzem Kugelschreiber, Füller etc.; Bleistift ist nicht erlaubt.

Zum **Bestehen** der Klausur sind **45** Punkte hinreichend.

☐ meine Note soll mit Matrikelnummer so bald wie möglich auf der Vorlesungs-Webseite veröffentlicht werden.

☐ meine Note soll nicht veröffentlicht werden; ich erfahre sie dann vom Prüfungsamt (bzw. für nicht von einem Prüfungsamt verwaltete Teilnehmer: beim Abholen des Scheins).

	Max. Punkte	Schätzung für "4"
Aufgabe 1 (XML-Entwurf)	30	20
Aufgabe 2 (XPath, XQuery)	40	24
Aufgabe 3 (XSLT)	20	6
Summe	90	50

Note:

Themenstellung: Fernbuslinien

Alle Klausuraufgaben basieren auf einem gemeinsamen "Auftrag": Seit 2011 ist die Einrichtung von *Fernbuslinien* in Deutschland erlaubt (wie sie bereits in vielen Ländern existieren). In der Klausur soll eine Datenbank eines Fernbuslinienbetreibers entworfen werden:

- Der Betreiber will mehrere Fernbuslinien einrichten. Die meisten Linien sammeln ihre Fahrgäste in einer Region schrittweise auf, fahren dann eine längere Strecke am Stück, und lassen die Fahrgäste in der Zielregion an einer oder mehreren Stationen wieder aussteigen.
- Jede Haltestelle hat einen eindeutigen Namen (meistens der Name der Stadt; z.B. "Frankfurt", aber auch "Frankfurt Flughafen"). Für jede Haltestelle ist der Name sowie die Strasse bzw. nähere Bezeichnung gespeichert (z.B. (Göttingen, ZOB Bahnhofsplatz)).

  Im den folgenden Aufgaben wird angenommen, dass "in XY" zu lesen ist als "an der Haltestelle, deren Name XY ist".
- Jede Linie hat eine eindeutige Nummer und besteht aus einer Liste der Haltestellen mit (relativen) Abfahrtszeiten und Fahrpreisen (jeweils bezogen auf den Startort; bei späterem Zustieg wird der entsprechende Teil des Preises abgezogen); vgl. Spezifikation der Linien 2460 (Flensburg-Hamburg-Frankfurt) und 2060 (Hamburg-Frankfurt) auf dem letzten Blatt der Klausur.
- Der Fahrplan gibt für jede Linie Wochentage und die *absoluten* Startzeiten am Startort an. Zum Beispiel fährt Linie 2460 werktags um 5:30, 8:00, 12:00 und 17:30 Uhr in Hamburg ab; samstags/sonntags aber um 8:00, 13:00 und 16:30 Uhr.
  Nehmen Sie an, dass jede Linie zumindest werktags einmal befahren wird.

Der konkrete Einsatzplan ordnet jeder durchzuführenden Fahrt (Tag, Zeit) einen Bus sowie einen Fahrer zu:

- Jeder Bus hat eine (eindeutige) Nummer.
- Jeder Fahrer hat einen Namen (Annahme: Vor- und Nachname sind zusammen eindeutig).
- Für jeden Tag (Datum) ist jeder an dem Tag durchzuführenden Fahrt ein Bus sowie ein Fahrer zugeordnet. So wird z.B. die erste Fahrt der Linie 2060 (Hamburg – Frankfurt) am 14.07.2011 um 05:30 von Hans Meier mit dem Bus 278 durchgeführt. Derselbe Bus fährt dann um 14:00 als Linie 6020 mit Fahrer Hans Meier wieder nach Hamburg.

Aufgabe 1 (XML-Entwurf [30 Punkte])
Die Datenspeicherung soll in XML stattfinden (nicht nur, weil das so in dieser Vorlesung behandelt wurde, sondern auch weil es für diesen Anwendungsfall ein sehr sinnvolles Datenformat ist; siehe auch Aufgabe 2d).

a) Geben Sie die DTD Ihrer Lösung an (15 P).

b) Geben Sie eine XML-Instanz an, die Ihre Lösung u.a. anhand der in der Spezifikation aufgeführten Beispieldaten illustriert. (10 P)

c) Geben Sie einen Automaten an, mit dem der Elementtyp validiert werden kann, in dem Sie die einzelnen Buslinien speichern. (5 P)

– hier DTD (und wenn man will XML) angeben –

Lösung Die Speicherung der Daten zu Haltestellen und die Zeiten/Preise der Linien, sowie die Zuordnung von Fahrern zu Touren ist relativ klar. Den Fahrplan, d.h. Linie/Wochentage/Zeiten kann man entweder zu Linie reinnehmen, oder separat ablegen (wie man es bei einer relationalen DB tun würde).

```
<!ELEMENT db (haltestelle+, linie+, fahrerzuordnung*)>
<!ELEMENT haltestelle EMPTY>
  <!ATTLIST haltestelle id ID #REQUIRED
                        name CDATA #REQUIRED
                        adresse CDATA #REQUIRED>
<!ELEMENT linie (stop,stop+, tour+)>
 <!ATTLIST linie nr CDATA #REQUIRED>
 <!ELEMENT stop EMPTY>
  <!ATTLIST stop haltestelle IDREF #REQUIRED
                 minuten CDATA #REQUIRED
                 preis   CDATA #REQUIRED>
 <!ELEMENT tour EMPTY>
  <!ATTLIST tour wochentag NMTOKEN #REQUIRED
                 abfahrt CDATA #REQUIRED>
<!ELEMENT zuordnung EMPTY>
  <!ATTLIST zuordnung linie CDATA #REQUIRED
                      datum CDATA #REQUIRED
                      zeit CDATA #REQUIRED
                      bus CDATA #REQUIRED
                      fahrer CDATA #REQUIRED>
```

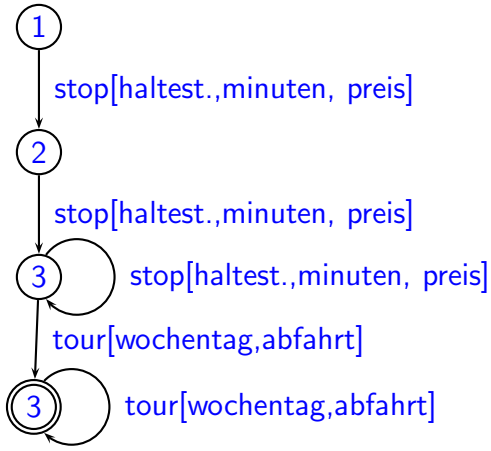
```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE db SYSTEM "buslinien.dtd">
<db>
 <haltestelle id="hambg" name="Hamburg" adresse="Hauptbahnhof"/>
 <haltestelle id="flensbg" name="Flensburg" adresse="ZOB"/>
 <haltestelle id="soltau" name="Soltau" adresse="Dorfplatz"/>
 <haltestelle id="hannover" name="Hannover" adresse="Hbf"/>
 <haltestelle id="goe" name="Goettingen" adresse="ZOB Bahnhofsplatz"/>
 <haltestelle id="kassel" name="Kassel" adresse="Karlsaue"/>
 <haltestelle id="ffm" name="Frankfurt" adresse="Bahnhofsvorplatz"/>
 <haltestelle id="fra" name="Frankfurt Flughafen" adresse="Terminal A Level 1"/>
 <haltestelle id="saarbr" name="Saarbruecken" adresse="Hbf"/>
 <haltestelle id="berlin" name="Berlin" adresse="Potsdamer Platz"/>

 <linie nr="2060">
  <stop haltestelle="hambg" minuten="0" preis="0"/>
  <stop haltestelle="soltau" minuten="50" preis="6.00"/>
  <stop haltestelle="hannover" minuten="100" preis="12.00"/>
  <stop haltestelle="goe" minuten="190" preis="20.00"/>
  <stop haltestelle="kassel" minuten="230" preis="25.00"/>
  <stop haltestelle="ffm" minuten="370" preis="39.00"/>
  <stop haltestelle="fra" minuten="390" preis="40.00"/>
  <tour wochentag="Werktag" abfahrt="05:30:00"/>
  <tour wochentag="Werktag" abfahrt="08:00:00"/>
  <tour wochentag="Werktag" abfahrt="14:00:00"/>
  <tour wochentag="SaSo" abfahrt="08:00:00"/>
  <tour wochentag="SaSo" abfahrt="13:00:00"/>
 </linie>
 <linie nr="6020">
  <stop haltestelle="fra" minuten="0" preis="0"/>
  <stop haltestelle="ffm" minuten="15" preis="1.00"/>
  <stop haltestelle="kassel" minuten="160" preis="15.00"/>
  <stop haltestelle="goe" minuten="200" preis="20.00"/>
  <stop haltestelle="hannover" minuten="290" preis="28.00"/>
  <stop haltestelle="soltau" minuten="340" preis="34.00"/>
  <stop haltestelle="hambg" minuten="390" preis="40.00"/>
  <tour wochentag="Werktag" abfahrt="09:00:00"/>
  <tour wochentag="Werktag" abfahrt="14:00:00"/>
  <tour wochentag="SaSo" abfahrt="10:00:00"/>
  <tour wochentag="SaSo" abfahrt="16:00:00"/>
 </linie>
 <linie nr="6610">
  <stop haltestelle="saarbr" minuten="0" preis="0"/>
  <stop haltestelle="fra" minuten="150" preis="20.00"/>
  <stop haltestelle="berlin" minuten="610" preis="65.00"/>
  <tour wochentag="Werktag" abfahrt="08:00:00"/>
  <tour wochentag="Werktag" abfahrt="10:00:00"/>
 </linie>

 <zuordnung linie="2060" datum="2011-14-07" zeit="05:30:00"
            bus="278" fahrer="Hans Meier"/>
 <zuordnung linie="6020" datum="2011-14-07" zeit="14:00:00"
            bus="278" fahrer="Hans Meier"/>
</db>
```

Alternative: <tour>nicht als subelement, sondern separat mit Kreuzreferenzattribut:

```
<!ELEMENT db (haltestelle+, linie+, tour+, fahrerzuordnung*)>
<!ELEMENT linie (stop,stop+)>
 <!ATTLIST linie nr CDATA #REQUIRED>
<!ELEMENT tour EMPTY>
  <!ATTLIST tour linie CDATA #REQUIRED
                 wochentag NMTOKEN #REQUIRED
                 abfahrt CDATA #REQUIRED>
```

@linie könnte man sich als ID/IDREF wünschen; allerdings sind dann reine Zahlwerte nicht zugelassen (wurde in der Klausur nicht als falsch bewertet), und man muss zusätzlich ids, z.B. "L2060" einführen.

c) Der Automat ist in diesem Fall relativ einfach, weil die Subelemente nur Attribute haben:



Aufgabe 2 (XPath, XQuery [40 Punkte])
Verwenden Sie für diese Aufgabe die von Ihnen entworfene XML-Datenbasis. Keine der Antworten soll Duplikate enthalten.

Hinweis (Syntax): Nehmen Sie für diese Aufgabe an, dass Sie mit dem Datentyp für "Zeit" rechnen (+ minuten, MAX, MIN) und vergleichen können (<, > etc.). Repräsentieren Sie Zeiten als Strings der Form "hh:mm" und Datumswerte als Strings der Form "YYYY-MM-DD". (Dies ist mit XML-Software auch (fast) genauso möglich.)
Hinweis (Anwendung): Nehmen Sie für diese Aufgabe an, dass folgendes gilt:

- es gibt keine Fahrten über Mitternacht,
- in jeder Linie kann an jeder Haltestelle ein- und ausgestiegen werden

a) Geben Sie eine XQuery-Anfrage an, die die Nummern aller Linien ausgibt, die werktags in Göttingen halten. (3 P)

Lösung

```
//linie[stop/id(@haltestelle)/@name="Goettingen" and
        tour[@wochentag = "Werktag"]]/string(@nr)
```

```
for $linie in //linie[stop/id(@haltestelle)/@name="Goettingen"]
where $linie/tour[@wochentag = "Werktag"]
return
  $linie/string(@nr)
```

(an dieser Stelle ist der Test auf Werktag eigentlich überflüssig, da vorausgesetzt wird, dass jede Linie werktags einmal fährt.)

b) Erweiterung von a) – Welche Linien halten werktags in Göttingen?
Für jede Fahrt soll die Liniennummer sowie die jeweilige Abfahrtszeit angegeben werden, und die Antworten sollen nach dieser Zeit geordnet sein, z.B.

```
<antwort linie="2060" zeit="11:10:00"/>
<antwort linie="6020" zeit="13:20:00"/>
<antwort linie="2060" zeit="16:10:00"/>
<antwort linie="6020" zeit="19:20:00"/>      (4 P)
```

Lösung

```
for $linie in //linie[stop/id(@haltestelle)/@name="Goettingen"],
    $tour in $linie/tour[@wochentag = "Werktag"]
let $minuten := $linie/stop[id(@haltestelle)/@name="Goettingen"]/@minuten,
    $time := xs:time($tour/@abfahrt) +
             xs:dayTimeDuration(concat("PT",$minuten,"M"))
    (: $tour/@abfahrt + $minuten :)
order by $time
return
 <antwort linie="{$linie/@nr}" zeit="{$time}"/>
```

c) Geben Sie eine HTML-Liste aller Linien mit ihren Strecken als Liste der Haltestellen in folgender Form aus: (4 P)

```
<ul>
  :
 <li> 2060
    <ul><li>Hamburg</li> <li>Soltau</li> ...
        <li>Frankfurt Flughafen</li><ul>
 </li>
 <li> 2061 <ul> ... </ul>
 </li>
  :
</ul>
```

Lösung

```
<ul>
 { for $linie in //linie
   order by number($linie/@nr)
   return <li> {string($linie/@nr)}
           <ul>
            { for $s in $linie/stop
              return
              <li> { $s/id(@haltestelle)/@name/string() } </li> }
           </ul>
          </li>
 }
</ul>
```

d) Warum ist XML für diese Anwendung wesentlich besser geeignet, als eine relationale Datenbank und SQL? (überlegen Sie sich, wie man Aufgabenteil (c) in SQL lösen müsste). (4 P)

Lösung XML-Daten sind geordnet. D.h., man kann die Haltestellen einfach der zeitlichen Abfolge nach ablegen, und in Dokumentordnung verarbeiten oder ausgeben. Mit einer SQL-Relation (deren Tupel ungeordnet sind), wie etwa auf dem letzten Blatt angegeben, müsste man schon zum Finden der Abfahrtshaltestelle einer Linie

```
SELECT *
FROM linie l1
WHERE minuten = (select min(minuten)
                 from linie l2
                 where l1.nr = l2.nr)
```

berechnen (anstatt `//linie/stop[1]`), und entsprechend dann für jede weitere.

e) Geben Sie eine Anfrage an, die die Nummern aller Linien ausgibt, mit denen man werktags ohne Umsteigen von Soltau nach Kassel fahren kann. (5 P)

Lösung

```
//linie[stop[id(@haltestelle)/@name="Soltau"
             and following-sibling::stop/id(@haltestelle)/@name = "Kassel"]
        and tour[@wochentag="Werktag"]]/string(@nr)
```

```
//linie[number(stop[id(@haltestelle)/@name="Soltau"]/@minuten)
        < number(stop[id(@haltestelle)/@name = "Kassel"]/@minuten)
        and tour[@wochentag="Werktag"]]/string(@nr)
```

f) Geben Sie eine XQuery-Anfrage an, die ausgibt, an welchem Ort der Fahrer Hans Meier am 14.7. seinen Dienst beendet. (5 P)

Lösung

```
//linie[@nr =
   //zuordnung[@datum="2011-14-07" and @fahrer="Hans Meier"
               and @zeit =
         max(//zuordnung[@datum="2011-14-07" and @fahrer="Hans Meier"]
                        /xs:time(@zeit))]
              /@linie]
  /stop[last()]/id(@haltestelle)/string(@name)
```

```
let $meierszuordnungen :=
   //zuordnung[@datum="2011-14-07" and @fahrer="Hans Meier"],
 $letzteZuordnung :=
   $meierszuordnungen[@zeit =
         max($meierszuordnungen/xs:time(@zeit))],
 $linie := //linie[@nr = $letzteZuordnung/@linie]
 return $linie/stop[last()]/id(@haltestelle)/string(@name)
```

```
let $zuordnungen :=
   ( for $z in //zuordnung[@datum="2011-14-07" and @fahrer="Hans Meier"]
     order by $z/xs:time(@zeit)
     return $z)
let $letzte := $zuordnungen[last()]
return //linie[@nr = $letzte/@linie]/stop[last()]/id(@haltestelle)/string(@name)
```

g) Geben Sie eine XQuery-Anfrage an, die ausgibt, welche Orte man werktags mit 1x umsteigen von Soltau aus erreichen kann. (10 P)

Lösung

```
for $linie1 in //linie[stop/id(@haltestelle)/@name="Soltau"],
    $tour1 in $linie1/tour[@wochentag="Werktag"],
    $stop0 in $linie1/stop[id(@haltestelle)/@name="Soltau"],
    $stop1 in $stop0/following-sibling::stop,
    $linie2 in //linie,
    $tour2 in $linie2/tour[@wochentag="Werktag"],
    $stop2 in $linie2/stop[@haltestelle = $stop1/@haltestelle]
let $zeit1 := xs:time($tour1/@abfahrt) +
              xs:dayTimeDuration(concat("PT",$stop1/@minuten,"M")),
    $zeit2 := xs:time($tour2/@abfahrt) +
              xs:dayTimeDuration(concat("PT",$stop2/@minuten,"M"))
where $zeit1 < $zeit2
return
  (for $stop3 in $stop2/following-sibling::stop
       [not (@haltestelle = $linie1/stop/@haltestelle)]
   return
     <verbindung
      ab="{xs:time($tour1/@abfahrt) +
             xs:dayTimeDuration(concat("PT",$stop0/@minuten,"M"))}"
      umsteigenIn="{$stop1/id(@haltestelle)/@name}"
      ankunftUmsteigen="{$zeit1}"
      abfahrtUmsteigen="{$zeit2}"
      ziel="{$stop3/id(@haltestelle)/@name}"
      ankunft = "{xs:time($tour2/@abfahrt) +
             xs:dayTimeDuration(concat("PT",$stop3/@minuten,"M"))}"/>)
```

h) **(Updates, abstrakte Sprachsyntax)**

Linie 2060 soll zusätzlich zwischen Hannover und Göttingen noch in Hildesheim (30 Minuten ab Hannover, 3.50E zusätzlich) halten.

Geben Sie ein Statement in einer Syntax an, die auf XQuery aufbaut (Sie können XQuery+Updates verwenden, können sich aber auch prinzipiell an der SQL-Syntax orientieren), das dieses Update auf Ihrem XML-Dokument sinnvoll spezifiziert. (5 P)

Lösung Die folgende Syntax ist nicht XQuery+Updates

```
update //linie[@nr="2060"]
insert
  <stop haltestelle="{//haltestelle[@name="Hildesheim"]/@id}"
        minuten="130" preis="15.50"/>
after stop[id(@haltestelle)/@name="Hannover"]
```

Als hypothetical Update in XQuery+Updates:

```
return
  insert
   <stop haltestelle="{//haltestelle[@name="Hildesheim"]/@id}"
         minuten="130" preis="15.50"/>
  after //linie[@nr="2060"]/stop[id(@haltestelle)/@name="Hannover"]
```

Aufgabe 3 (XSLT [20 Punkte])
Geben Sie ein XSLT-Stylesheet an, das eine XML-to-HTML-Transformation ausführt, das für jede Linie den Fahrplan in der folgenden Form als HTML-Tabelle ausgibt:
(Erzeugen Sie die richtige Struktur; Zentrierung, Linien etc. sind unwichtig. Wenn Sie sich bei bestimmten XSL-Kommandos nicht sicher sind, "erfinden" Sie eine *sinnvolle* Syntax.)

2060	Preis			
Hamburg	0	5:30	8:00	14:00
Soltau	6.00	6:20	8:50	14:50
Hannover	12.00	7:10	9:40	15:40
Göttingen	20.00	8:40	11:10	17:10
Kassel	25.00	9:20	11:50	17:50
Gießen	33.00	10:50	13:20	19:20
Frankfurt	39.00	11:40	14:10	20:10
Frankfurt Flughafen	40.00	12:00	14:30	20:30

Betrachten Sie dabei **nur den Werktags-Fahrplan**.

Lösung

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                xmlns:xs="http://www.w3.org/2001/XMLSchema"
                version="2.0">
<xsl:output method="html" indent="yes"/>
<xsl:template match="db">
  <xsl:apply-templates select="linie"/>
</xsl:template>
<xsl:template match="linie">
 <table>
  <tr> <td> <xsl:value-of select="@nr"/> </td>
       <td> Preis </td>
  </tr>
  <xsl:for-each select="stop">
   <xsl:variable name="minuten" select="@minuten"/>
     <!-- minuten muss man in einer Variable festhalten -->
   <tr>
    <td> <xsl:value-of select="id(@haltestelle)/@name"/></td>
    <td> <xsl:value-of select="@preis"/></td>
    <xsl:for-each select="../tour[@wochentag='Werktag']">
     <td>
      <xsl:value-of select="xs:time(@abfahrt)
                    + xs:dayTimeDuration(concat('PT',$minuten,'M'))"/>
      <!-- <xsl:value-of select="@abfahrt + $minuten"/> -->
     </td>
    </xsl:for-each>
   </tr>
  </xsl:for-each>
 </table>
</xsl:template>
</xsl:stylesheet>
```

Anmerkung: auch hier nutzt man es aus, dass die einzelnen Fahrten einer Linie in XML zeitlich geordnet sind und man sie einfach der Reihe nach durchgehen kann, um die Spalten zu erzeugen.

[Trennen Sie dieses Blatt am besten vor Beginn der Bearbeitung ab]

Die folgenden Basisdaten für die Linie 2460 (Flensburg nach Frankfurt), 2060 (Hamburg nach Frankfurt mit mehreren Zwischenstops), und 6020 sollen (unter anderem) gespeichert werden:

Linie			
Nr	**Ort**	**Minuten**	**Preis**
:	:	:	:
2460	Flensburg	0	0.00
2460	Schleswig	25	2.00
2460	Rendsburg	55	4.00
2460	Neumünster	85	6.00
2460	Quickborn	120	9.00
2460	Pinneberg	135	10.00
2460	Hamburg	155	12.00
2460	Frankfurt	615	42.00
2460	Frankfurt Flughafen	640	43.00
2060	Hamburg	0	0.00
2060	Soltau	50	6.00
2060	Hannover	100	12.00
2060	Göttingen	190	20.00
2060	Kassel	230	25.00
2060	Gießen	320	33.00
2060	Frankfurt	370	39.00
2060	Frankfurt Flughafen	390	40.00
6020	Frankfurt Flughafen	00	0.00
6020	Frankfurt	20	1.00
:	:	:	:
6020	Hamburg	390	40.00
:	:	:	:

Z.B. Linie 2060 ist 190 Minuten nachdem der Bus in Hamburg losgefahren ist, in Göttingen. Bis dahin kostet die Fahrt 20.00E.