Klausur "Semistrukturierte Daten und XML"
Sommersemester 2004
Prof. Dr. Wolfgang May
27. Juli 2004, 10-12 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 aus Munopag (bzw. für nicht im Munopag geführte Studierende: beim Abholen des Scheins).

	Max. Punkte	Schätzung für "4"
Aufgabe 1 (Allgemeines)	11	7
Aufgabe 2 (XML, XPath)	20	15
Aufgabe 3 (XQuery)	22	12
Aufgabe 4 (XSLT)	18	8
Aufgabe 5 (Verschiedenes)	19	8
Summe	90	50

Note:

Aufgabe 1 (Allgemeines [11 Punkte])
(alle Fragen können kurz und stichpunktartig beantwortet werden)

1. Welche "Fortschritte" bedeutete die "Erfindung" des relationalen Datenmodells, der relationalen Algebra und SQL gegenüber den vorherigen, auf dem Netzwerkmodell basierenden Datenbanken? (4 P)

2. Was bedeutet "Abgeschlossenheit" einer Anfragesprache? (3 P)

3. Welche Funktionalitäten und Grundlagen haben XML-Datenbanken und relationale (und alle anderen) Datenbanken gemeinsam? (4 P)

Lösung

1. Relationales Modell/Algebra: mathematisch fundierte Grundlage (ermöglicht Äquivalenzbegriff von Abfragen und algebraische Optimierung)
   SQL: deklarative mengenorientierte Anfragesprache, knappe und klare, einfach verständliche (auch für Nicht-Informatiker) Syntax

2. Das Ergebnis einer Anfrage ist wieder eine Struktur des zugrundeliegenden Modells (eine Relation, ein XML-Baum etc.)
   [Der Begriff der Orthogonalität hängt eng damit zusammen: Orthogonalität bedeutet, dass man Konstrukte beliebig schachteln kann (und damit eine "funktionale" Sprachstruktur erreicht), dies ist nur möglich, wenn das Ergebnis des inneren Ausdrucks "passt"].

3. Persistente Speicherung, Mehrbenutzerbetrieb, Transaktionskonzepte, Sicherheit (Zugriffssicherheit und Datensicherheit); oft auch interne Speicherzugriffsmechanismen (Indexstrukturen, Caching). Alles was "Datenbanken" so ausmacht ...

Aufgabe 2 (XML, XPath [20 Punkte])
Gegeben ist das folgende XML-Fragment (gegenüber dem Original-Hamlet der Vorlesung leicht veränderte DTD):

```
<?xml version="1.0"?>
<!DOCTYPE PLAY [
  <!ELEMENT PLAY (TITLE, PERSON+, SCENE+)>
  <!ELEMENT TITLE (#PCDATA)>
  <!ELEMENT PERSON EMPTY>
    <!ATTLIST PERSON NAME ID #REQUIRED
                     DESCRIPTION CDATA #IMPLIED>
  <!ELEMENT SCENE (SPEECH+)>
  <!ELEMENT SPEECH (LINE+)>
    <!ATTLIST SPEECH SPEAKER IDREF #REQUIRED>
  <!ELEMENT LINE ANY>
  <!ELEMENT LOUD (#PCDATA)>
  ]
<PLAY>
 <TITLE>The Tragedy of Hamlet, Prince of Denmark</TITLE>

 <PERSON NAME="Claudius" DESCRIPTION="King of Denmark"/>
 <PERSON NAME="Hamlet" DESCRIPTION="Nephew to the present king"/>
 <PERSON NAME="Polonius" DESCRIPTION="Lord chamberlain"/>
 <PERSON NAME="Ophelia" DESCRIPTION="Daughter to Polonius"/>
   :
 <SCENE>
   <SPEECH SPEAKER="Hamlet">
     <LINE><LOUD>To be, or not to be</LOUD>: that is the question:</LINE>
     <LINE>Whether 'tis nobler in the mind to suffer</LINE>
      :
     <LINE>The fair Ophelia! Nymph, in thy orisons</LINE>
     <LINE>Be all my sins remember'd.</LINE>
   </SPEECH>
   <SPEECH SPEAKER="Ophelia">
     <LINE><LOUD>Good my lord</LOUD>,</LINE>
     <LINE>How does your honour for this many a day?</LINE>
   </SPEECH>
   <SPEECH SPEAKER="Hamlet">
     <LINE>I humbly thank you; well, well, well.</LINE>
   </SPEECH>
    :
 </SCENE>
</PLAY>
```

1. Geben Sie einen XPath-Ausdruck an, der die Beschreibung der Person "Claudius" zurückgibt. (2 P)

2. Geben Sie einen XPath-Ausdruck an, der alle Textstellen, die <LOUD> gesprochen werden, zurückgibt. (1 P)

3. Geben Sie einen XPath-Ausdruck an, der die Menge der Namen aller Personen zurückgibt, die etwas <LOUD> sprechen. (3 P)

4. Betrachten Sie den XPath-Ausdruck

```
/descendant::SPEECH[contains(line/text(),"To be or not to be")]
   /following-sibling::SPEECH[1]/@SPEAKER
```

   Geben Sie an, welche(n) Knoten der Ausdruck als Ergebnis hat (2 P).

5. Geben Sie einen XPath-Ausdruck an, der die Beschreibung der Person zurückgibt, die auf Hamlets Auftritt, in dem er "To be or not to be" sagt, antwortet. (3 P)

6. Erklären Sie anhand des Ausdrucks aus Teil (4.) die Begriffe *Achse*, *Vorwärts- und Rückwärtsachse* (Sie können den Begriff "Kontextknoten" ohne separate Erklärung voraussetzen; 6 P). Geben Sie für die letzteren jeweils zwei Beispiele an (je 1.5 P).

Lösung

1. `//PERSON[@NAME="Claudius"]/@DESCRIPTION` – genauer: man müsste eigentlich sogar value(...) anwenden, um wirklich den Wert, und nicht den ganzen Attributknoten zu erhalten.

2. `//LOUD/text()`
   (0.5 P Abzug, wenn text() fehlt)

3. Viele Möglichkeiten:

```
id(//LOUD/ancestor::SPEECH/@SPEAKER)/@NAME
id(//SPEECH[./LOUD]/@SPEAKER)/@NAME
//PERSON[@NAME=//LOUD/ancestor::SPEECH/@SPEAKER]/@NAME
//PERSON[@NAME=//SPEECH[./LOUD]/@SPEAKER]/@NAME
```

   Hinweis: `//LOUD/ancestor::SPEECH/@SPEAKER` gibt nicht die Menge der Namen, sonden ggf. jeden Namen mehrmals.
   (1 P Abzug, falls Duplikate nicht eliminiert werden)
   Noch ein Hinweis: "Menge der ..." ist nicht "Anzahl der ...", sondern bedeutet, dass Duplikate zu eliminieren sind.

4. einen Attributknoten mit Name "SPEAKER" und Wert "Ophelia"; die Schreibweise `SPEAKER='Ophelia'` ist auch richtig.

5. Der gesuchte Ausdruck ist z.B.

```
id(//SPEECH[contains(line/text(),``To be or not to be'')]
 /following-sibling::SPEECH[1]/@SPEAKER)/@DESCRIPTION
//SPEECH[contains(line/text(),``To be or not to be'')]
 /following-sibling::SPEECH[1]/id(@SPEAKER)/@DESCRIPTION
```

   (die Abfrage `@SPEAKER='Hamlet'` kann noch hinzugefügt werden)

6. Verschiedene Möglichkeiten:

   (a) XPath-orientiert: Jeder XPath-Schritt setzt sich aus /Achse::Knotentest[Filter]/ zusammen. Das Ergebnis jedes Navigationsschrittes in XPath ist eine Menge von Knoten. Zur Auswertung des nächsten Schrittes wird jeweils einer dieser Knoten als Kontextknoten ausgewählt. Die Achse gibt dazu an, welche Knoten ausgehend vom Kontextknoten betrachtet werden, z.B. alle seine Kinder (child::), alle seine Attribute (attribute::) oder seine nachfolgenden Geschwister (following-sibling::). Vorwärtsachsen: z.B. child und following-sibling. Rückwärtsachsen: ancestor, parent, preceding-sibling.

   (b) XML-Baum-orientiert: ausgehend von einem beliebigen Elementknoten kann man entlang der Achsen (virtuell) Knoten des Baumes aufzählen. Die child-Achse enthält z.B. alle Kinder (in Dokumentordnung – eine Vorwärtsachse). Die following-sibling-Achse enthält alle nachfolgenden Geschwister (auch in Dokument-Ordung). Die parent-Achse enthält nur das eindeutige Elternelement, die ancestor-Achse enthält alle Elemente auf dem Pfad zur Wurzel (entgegen der Dokumentordnung – eine Rückwärtsachse). Die preceding-sibling-Achse zählt alle vorhergehenden Geschwister entgegen der Dokumentordnung auf (Rückwärtsachse).

Für die beiden folgenden Aufgaben (XQuery und XSLT) sei das folgende XML-Dokument gegeben:

```
<Munopag>
  <Dozent Name="Mueller"/>
  :
  <Student MatNo="S1" Name="Hans Schmidt"/>
  :
  <Veranstaltung Name="Informatik I">
    <Klausur Semester="WS2000" Dozent="Meier">
      <bestanden MatNo="S1">3.7</bestanden>
      <nichtbestanden MatNo="S2"/>
      <bestanden MatNo="S3">2.0</bestanden>
      <nichtbestanden MatNo="S4"/>
    </Klausur>
    <Klausur Semester="WS2001" Dozent="Mueller">
      <bestanden MatNo="S2"/>3.0</bestanden>
      <nichtbestanden MatNo="S4"/>
      <bestanden MatNo="S5">1.3</bestanden>
    </Klausur>
    :
  </Veranstaltung>
  <Veranstaltung Name="Informatik II">
    :
  </Veranstaltung>
</Munopag>
```

Namen und Matrikelnummern seien jeweils in vernuenftiger Weise als ID und IDREF deklariert.

Aufgabe 3 (XQuery [22 Punkte])
Geben Sie für die Anfragen (2)-(5) je eine XQuery-Anfrage an, die das Ergebnis in dem jeweils angegebenen Format ausgibt.
[Hinweis: bestandene Prüfungen können nicht wiederholt werden!]

1. Beschreiben Sie kurz, was XPath und XQuery miteinander zu tun haben, und was XQuery kann, was XPath nicht kann. (5 P)

2. Geben Sie alle Matrikelnummern von Studenten an, die "Informatik I" irgendwann bestanden haben (3 P):
   <Student MatNo="*matno*"/>

3. Geben Sie alle Paare von Matrikelnummern von Studenten bzw. Namen von Dozenten an, so dass der Student die "Informatik I" bei dem Dozenten bestanden hat (3 P):
   <Ergebnis Student="*matno*" Dozent="*doz-name*"/>

4. Geben Sie die Menge aller Paare (MatrNo eines Studenten, Name eines Dozenten) an, so dass der Student bei dem Dozenten in irgendeiner Prüfung nicht bestanden hat (Format beliebig, z.B. wie oben) (5 P).

5. Geben Sie für jeden Studenten ein Zeugnis in Form einer Liste der bestandenen Fächer mit Note aus (6 P):
   <Zeugnis MatNo="*matno*">
   <Fach Name="$Veranstaltung_1$" Note="$Note_1$"/>
   <Fach Name="$Veranstaltung_2$" Note="$Note_2$"/>
   :
   </Zeugnis>

Lösung

1. XPath wird in XQuery als Adressierungssprache verwendet. XQuery ist, wie SQL, eine deklarative, klauselbasierte Anfragesprache. Mit XQuery sind auch Joins sowie die Erzeugung von Ergebnisstrukturen möglich.

2.
```
for $x in //Veranstaltung[@Name="Informatik I"]//bestanden
return <Student MatNo="{$x/@MatNo}"/>
```

3.
```
for $x in //Veranstaltung[@Name="Informatik I"]//bestanden
return <Ergebnis Student={$x/@MatNo}
                 Dozent={$x/ancestor::Klausur/@Dozent}/>

for $k in //Veranstaltung[@Name="Informatik I"]/Klausur
for $x in $k//bestanden
return <Ergebnis Student={$x/@MatNo}
                 Dozent={$k/@Dozent}/>
```

   Hinweis: auch mit FOR-Join und WHERE (wie im nächsten Aufgabenteil) möglich.

4.
```
for $d in //Dozent, $s in //Student
where some $nb in //nichtbestanden
    satisfies ($nb/@MatNo=$s/@MatNo and
               $nb/ancestor::Klausur/@Dozent = $d/@Name)
return <Ergebnis Student="{$s/@MatNo}"
                 Dozent="{$d/@Name}"/>
```

Hinweis: mit `for $nb in //nichtbestanden ...` wären Duplikate enthalten.

5.
```
for $s in //Student
return
 <Zeugnis MatNo="{$s/@MatNo}">
   { for $best in //bestanden[@MatNo=$s/MatNo]
     return
        <Fach Name="{$best/ancestor::Veranstaltung/@Name}"
              Note="{$best/text()}/>
   }
 </Zeugnis>
```

oder

```
for $s in //Student
let $faecher :=
    for $best in //bestanden[@MatNo=$s/@MatNo]
        return
        <Fach Name="{$best/ancestor::Veranstaltung/@Name}"
              Note="{$best/text()}/>
return
 <Zeugnis MatNo="{$s/@MatNo}">
   $faecher
 </Zeugnis>
```

Aufgabe 4 (XSLT [18 Punkte])
Diese Aufgabe verwendet den Teil "Veranstaltungen" aus dem angebenen Munopag-Beispiel.

1. Beschreiben Sie kurz, wie `<xsl:template match='pattern'>` und `<xsl:apply-templates select='xpath-ausdruck'>` zusammenarbeiten. Sie können hierzu das in der nächsten Teilaufgabe zu erstellende Beispiel verwenden. (5 P)

2. Geben Sie Templates für ein XSLT-Stylesheet an, das für eine gegebenes Klausur-Element (z.B. Informatik I in 2001) eine Notentabelle in XHTML wie folgt (wie in der Eingabe nach Matrikelnummern geordnet) ausdruckt (10 P):

   - Kopfzeile: Name der Veranstaltung sowie Semester,
   - Je eine Zeile für jeden Teilnehmer mit den Spalten "Matrikelnummer", "Note" (nicht bestanden wird mit 5.0 bewertet), sowie ein "*", falls der Teilnehmer nicht bestanden hat.

   Der Aufruf der von Ihnen zu schreibenden Templates erfolgt z.B. durch

```
<xsl:template match="/">
 <xsl:apply-templates
  select="//Veranstaltung[@Name='Informatik I']
                         /Klausur[@Semester='WS2001']"/>
</xsl:template>
```

3. Was müssen Sie ändern, wenn zuerst alle "bestandenen" Teilnehmer, und dann alle "nicht bestandenen" Teilnehmer aufgezählt werden sollen? (3 P)

Lösung
Apply-templates wählt durch den als select-Attribut gegebenen XPath-Ausdruck an einem gegebenen Punkt aus, welche Knoten an dieser Stelle ein Teilergebnis beitragen sollen. Für diese Knoten wird dann jeweils das passende xsl:template (d.h., das, dessen match-Attribut zutrifft) angewendet.

```
<xsl:template match="Klausur">
  <table>
   <tr colspan="3">
    <td>
     <xsl:value-of select="./parent::Veranstaltung/@name"/>
     <xsl:value-of select="@semester"/>
    <td>
   </tr>
   <xsl:apply-templates>
         <!-- oder mit select="bestanden|nichtbestanden"
              bzw select="*" -->
```

```
    </table>
  </xsl:template>
  <xsl:template match="bestanden">
    <tr>
      <td><xsl:value-of select="@MatNo"/></td>
      <td><xsl:value-of select="text()"/></td>
      <td></td>  <!-- kann man auch weglassen -->
    </tr>
  </xsl:template>
  <xsl:template match="nichtbestanden">
    <tr>
      <td><xsl:value-of select="@MatNo"/></td>
      <td>5.0</td>
      <td>*</td>
    </tr>
  </xsl:template>
```

Alternative mit xsl:for-each (da man aber zwischen bestanden und nicht bestanden dann innen unterscheiden muss, ist das etwas umständlicher:

```
  <xsl:template match="Klausur">
    <table>
     <tr colspan="3">
      <td>
       <xsl:value-of select="./parent::Veranstaltung/@name"/>
       <xsl:value-of select="@semester"/>
      <td>
     </tr>
     <xsl:for-each select="bestanden|nichtbestanden">
      <tr>
       <td><xsl:value-of select="@MatNo"/></td>
       <xsl:if test="text()">
         <td><xsl:value-of select="."></td><td></td>
       </xsl:if>
       <xsl:if test="not text()">
         <td>5.0</td><td>*</td>
       </xsl:if>
      </tr>
     </xsl:for-each>
    </table>
  </xsl:template>
```

Änderung der Aufzählung:

```
     <xsl:apply-templates select="bestanden"/>
     <xsl:apply-templates select="nichtbestanden"/>
```

bzw.

```
<xsl:for-each select="bestanden">...</>
<xsl:for-each select="nichtbestanden">...</>
```

(dann benötigt man das innere xsl:if nicht mehr).

Aufgabe 5 (Verschiedenes [19 Punkte])

1. Beschreiben Sie kurz, wie eigene SimpleTypes in **XML Schema** aus existierenden Typen (z.B. xsd:string oder xsd:float) abgeleitet werden, und was man damit machen kann. Geben Sie ein Beispiel eines solchen Datentypen (mit XML Schema Quellcode). (5 P)

2. Gegeben sei die folgende Relation einer relationalen Datenbank:

Buch			
Titel	Verlag	ISBN	Preis
"XML"	"Springer"	0-815	50.00
"SQL"	"Hanser"	47-11	64.00

   Geben Sie an, wie diese Tabelle *generisch* in eine XML-Repräsentation abgebildet werden kann (kurze textuelle Beschreibung der Grundideen Ihrer Abbildung, XML-Ergebnis; 6 P); geben Sie auch die DTD dazu an (4 P).

3. Angenommen, man hat einen **XLink**-fähigen Browser. Welches Verhalten zeigt eine Seite, die das folgende XHTML-Fragment enthält: (4 P)

```
<b>Die folgenden Teilgebiete werden in der Vorlesung behandelt</b>
<ul><li> ... </li>
    <li>
     <b xlink:type="simple"
        xlink:actuate="OnRequest"
        xlink:show="embed"
        xlink:href="http://ap34.inf.uni-goe.de/index.html/p[a/@name='XSLT']">
      XSLT</b>
  </li>
   :
</ul>
```

Lösung

1. Einschränkung eines existierenden SimpleType, z.B. durch Wertebereichseinschränkung (geogr. Länge aus Float) oder Längenbeschränkung (car-codes als Strings von 1-3 Zeichen Länge). Verwendet werden diese Datentypen als Attributtypen oder für Textinhalt von Elementen. Beispiel (siehe auch Vorlesungsfolien):

```
<xs:simpleType name="mondial:longitude"
      <xs:restriction base="xs:float">
          <xs:minExclusive value="-180" />
          <xs:maxInclusive value="180" />
      </xs:restriction>
</xs:simpleType>
```

2. siehe Vorlesungsfolien (Abbildung relationaler Daten nach XML). Beispiel (die ganz generische Methode):

```
<row name="buch">
  <column name="Titel">XML</column>
  <column name="Verlag">Springer</column>
  <column name="ISBN">0-815</column>
  <column name="Preis">50.00</column>
</row>
<row name="buch">
  <column name="Titel">SQL</column>
  <column name="Verlag">Hanser</column>
  <column name="ISBN">47-11</column>
  <column name="Preis">64.00</column>
</row>

<!ELEMENT row (column+)>
   <!ATTLIST row name #REQUIRED CDATA>
<!ELEMENT column (#PCDATA)>
   <!ATTLIST row name #REQUIRED CDATA>
```

3. Laden: Die Texte der Listenelemente, hier für den "XSLT"-Eintrag gezeigt, sind nicht nur fett gedruckt, sondern auch als "clickable" markiert (aufgrund der xlink:type="simple"-Angabe).
   Klicken: Wenn der Benutzer darauf klickt, wird der mit dem entsprechenden Anker versehene Paragraph – hier `<p><a name="XSLT">...</p>` – aus der index.html-Webseite von der ap34 an dieser Stelle in die Tabelle eingebettet.