

PRÜFUNG IN "SEMI-STRUKTURIERTE DATEN" 184.705			25. 6. 2019
Kennnr.	Matrikelnr.	Familienname	Vorname

Arbeitszeit: 100 Minuten.

Aufgaben sind auf den Angabeblättern zu lösen; Zusatzblätter werden nicht gewertet.

Bitte tragen Sie Namen, Studienkennzahl und Matrikelnummer ein. Halten Sie Ihren Studentenausweis bereit.

Aufgabe 1:

(12)

Betrachten Sie folgende XML Schema Datei **test.xsd**:

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">

  <xsd:element name="W" type="newtype"/>

  <xsd:complexType name="newtype">
    <xsd:choice>
      <xsd:sequence>
        <xsd:element name="Z" type="xsd:NMTOKEN"/>
        <xsd:element name="W" minOccurs="0" maxOccurs="2" type="newtype"/>
      </xsd:sequence>
      <xsd:sequence>
        <xsd:element name="W" type="newtype"/>
        <xsd:element name="Z" type="xsd:int"/>
      </xsd:sequence>
    </xsd:choice>
  </xsd:complexType>

</xsd:schema>
```

Betrachten Sie weiters die acht verschiedenen XML-Dateien, die unten angeführt sind.

Sie können davon ausgehen, dass alle folgenden XML-Dateien wohlgeformt sind. Es geht also lediglich darum, ihre Gültigkeit bezüglich test.xsd zu entscheiden.

Kreuzen Sie an, welche der folgenden xml-Dateien gültig bezüglich test.xsd sind.

- | | | |
|--|---|---|
| 1. <W><Z>SSD</Z></W> | valid <input checked="" type="checkbox"/> | invalid <input type="checkbox"/> |
| 2. <W><Z>Einstein</Z><W><Z>A</Z></W></W> | valid <input checked="" type="checkbox"/> | invalid <input type="checkbox"/> |
| 3. <W><Z>Emmy Noether</Z></W> | valid <input type="checkbox"/> | invalid <input checked="" type="checkbox"/> |
| 4. <W><Z>Albert</Z><Z>1</Z></W> | valid <input type="checkbox"/> | invalid <input checked="" type="checkbox"/> |
| 5. <W><W><W><Z>ADBS</Z></W><Z>007</Z></W></W> | valid <input type="checkbox"/> | invalid <input checked="" type="checkbox"/> |
| 6. <W><W><W><Z>ADBS</Z></W><Z>100</Z></W><Z>42</Z></W> | valid <input checked="" type="checkbox"/> | invalid <input type="checkbox"/> |

(Für jede korrekte Antwort 1.5 Punkte, für jede falsche Antwort -1.5 Punkte, unbeantwortete Fragen 0 Punkt, Insgesamt nicht weniger als 0 Punkte)

Aufgabe 2:

(15)

Beantworten Sie, die folgenden Fragen kurz und bündig (Für jede korrekte Antwort 1.5 Punkte).

1. Erläutern Sie kurz den Unterschied zwischen `let` und `for` statements in XQuery.

Antwort: Mit `for` wird jedes Element einer Sequenz einzeln an die Variable gebunden (es wird über die Elemente der Sequenz iteriert). Mit `let` wird die vollständige Sequenz auf einmal an die Variable gebunden.

2. Wie verhält sich der Speicherbedarf von Tree-based Parsern wie DOM zur Größe des geparsen XML Dokuments?

Antwort: Der Speicherbedarf steigt linear mit der Größe des geparsen XML Dokuments.

3. Was macht das Default-Template in XSLT für Elemente?

Antwort: Es wendet die Templates der Kinder an.

4. Wie geht XSLT mit der Situation um wenn mehrere Templates auf ein Element matchen?

Antwort: Es wird nur eines ausgeführt, dabei wird das spezifischste Template ausgewählt.

5. Welches Datenmodell verwenden wir für semistrukturierte Daten?

Antwort: Bäume mit Labels auf den Kanten.

6. Auf welche Teile eines XML Dokuments wird der Default Namespace angewendet?

Antwort: Auf Elemente ohne Prefix.

7. Geben Sie einen Formalismus (ein Format) an um semistrukturierte Daten zu speichern:

Antwort: Object Exchange Model (OEM), JavaScript Object Notation (JSON), eXtensible Markup Language (XML)

8. Wir haben verschiedene Sprachen um Schemata zu definieren kennen gelernt. Welche hat die größte Ausdrucksstärke?

Antwort: XML Schema

9. Zu welchem Zweck werden Namespaces verwendet?

Antwort: Um Elemente/Attribute mit gleichem Namen aus unterschiedlichen Quellen zu unterscheiden. Um Elemente/Attribute zu gruppieren.

10. Wieso können URIs nicht direkt als prefix für XML Elemente/Attribute verwendet werden?

Antwort: Die resultierenden Elementnamen sind keine gültigen XML-Namen.

Die folgenden Aufgaben 3 – 7 beziehen sich auf das XML-Dokument `auctions.xml`, das Sie auf der letzten Seite dieser Prüfungsangabe finden.

Aufgabe 3:

(12)

Vervollständigen Sie das DTD Dokument `auctions.dtd`, sodass XML-Dokumente in der Gestalt von `auctions.xml` (siehe Anhang) bezüglich dieser DTD gültig sind. Berücksichtigen Sie beim Erstellen der DTD folgende Punkte:

- Ein `product` Element enthält ein `productname` Element, möglicherweise ein `description` Element, und beliebig viele `category` Elemente (in dieser Reihenfolge). Zusätzlich enthält es immer zwei Attribute: eine eindeutige id `ID` und ein Attribute `auctionEnd`.
- Das `description` Element enthält eine textuelle Beschreibung und kann Referenzen mittels `a` Elementen enthalten. `a` Elemente enthalten immer ein Attribut `href`.
- Im `categories` Element können beliebig viele `category` Elemente gespeichert sein.
- `category` Elemente treten in zwei Formen auf. Als Kind des `categories` Elements enthält es eine Bezeichnung und ein Attribut `id` als eindeutige id. Als Kind eines `product` Elements ist es leer und hat ein Attribut `ref` das auf die id eines unter `categories` gespeicherten `category` Elements verweist.
- Das `user` Element enthält beliebig viele `user` Elemente. Jedes `user` Element enthält entweder ein `fullname` oder ein `firstname` und ein `lastname` Element. Zusätzlich gibt es immer in eindeutiges `username` Attribut.
- Das `bids` Element enthält beliebig viele `bidproduct` Elemente. Jedes `bidproduct` enthält mindestens ein `bid` Element und eine Attribut `id` das auf ein `product` Element verweist. `bid` Elemente speichern eine ganze Zahl und ein Attribut `user` das auf den usernamen eines users verweisen.
- Wenn nicht angegeben treffen Sie plausible Annahmen über Typen von Attributen und Elementen.

File `auctions.dtd`:

```
<!ELEMENT auctions (products, categories, users, bids)>
<!ELEMENT products (product*)>

<!ELEMENT product (productname, description?, category*)>
<!ATTLIST product id ID #REQUIRED auctionEnd CDATA #REQUIRED>

<!ELEMENT productname (#PCDATA)>
<!ELEMENT description (#PCDATA | a)*>
<!ELEMENT a (#PCDATA)>
<!ATTLIST a href CDATA #REQUIRED>

<!ELEMENT categories (category*)>
<!ELEMENT category (#PCDATA)>
<!ATTLIST category id ID #IMPLIED ref IDREF #IMPLIED>

<!ELEMENT users (user*)>

<!ELEMENT user (fullname|(firstname,lastname))>
<!ATTLIST user username ID #REQUIRED>
<!ELEMENT fullname (#PCDATA)>
<!ELEMENT firstname (#PCDATA)>
<!ELEMENT lastname (#PCDATA)>

<!ELEMENT bids (bidproduct*)>

<!ELEMENT bidproduct (bid+)>
<!ATTLIST bidproduct id IDREF #REQUIRED>
<!ELEMENT bid (#PCDATA)>
<!ATTLIST bid user IDREF #REQUIRED>
```

Aufgabe 4:

(10)

Betrachten Sie die folgenden XPath-Abfragen angewandt auf das Dokument **auctions.xml** (siehe Anhang).

- Falls der angegebene XPath Ausdruck keine Knoten selektiert, notieren Sie im entsprechenden Feld "leere Ausgabe".
- Falls als Ergebnis eine Zahl selektiert wird (count, sum, ...), geben Sie diese Zahl an.

Geben Sie nun die entsprechende Ausgaben der folgenden XPath-Abfragen an.

```
count(//categories/category)
```

3

```
//product[./category/@ref=//categories/category[text()='book']/@id]/@id
```

p1
p3

```
//product[category/@ref="cat1"][2]/productname
```

<productname>XML in a Nutshell</productname>

```
//product[2][category/@ref="cat1"]/productname
```

leere Ausgabe

```
//bids/bidproduct[sum(bid)>499]/@id
```

p1
p2

Aufgabe 5:

(10)

Ergänzen Sie die Java Klasse auf dieser und der nächsten Seite so, dass sie beim Aufruf, mit XML files die dem Schema der gegebenen **auctions.xml** folgen, den folgenden Output erzeugt:

- Für jeden User soll eine Zeile ausgegeben werden in der die **Summe** aller Gebote des Users zusammen mit dem username steht.
- Halten Sie sich bei der Formatierung an den Beispiel Output.
- Hat ein user nie ein Gebot abgegeben, so muss auch keine Zeile für ihn ausgegeben werden.

Beispiel Output beim Aufruf mit der gegebenen **auctions.xml**:

```
Homer bid 200 in total
Bart bid 1300 in total
Lisa bid 410 in total
```

(Reihenfolge der user undefiniert)

Vervollständigen Sie den folgenden Code:

```
public class RunSAX extends DefaultHandler {
    private String bidUser, eleText;
    private HashMap<String, Integer> total = new HashMap<String, Integer>();

    public void characters(char[] text, int start, int length)
    throws SAXException {
        eleText = new String(text, start, length);
    }

    public void startElement(String namespaceURI, String localName,
        String qName, Attributes atts) throws SAXException {
        if ("bid".equals(localName))
            bidUser = atts.getValue("user");
    }

    public void endElement(String namespaceURI, String localName,
        String qName) throws SAXException {
        if ("bid".equals(localName)){
            Integer old = total.getOrDefault(bidUser, 0);
            Integer n = Integer.parseInt(eleText);
            total.put(bidUser, old + n);
        }
    }

    public void endDocument() throws SAXException {
        for(String u : total.keySet())
            System.out.println(u + " bid " + total.get(u) + " in total");
    }

    public static void main(String[] args) throws Exception {
        InputSource source = new InputSource(new FileInputStream("auction.xml"));

        XMLReader xr = XMLReaderFactory.createXMLReader();
        RunSAX rs = new RunSAX();
        xr.setContentHandler(rs);

        xr.parse(source);
    }
}
```

Betrachten Sie folgende XQuery `auctions.xq`:

```
<actn>
{
  for $p in doc("auctions.xml")//product
  order by $p/@auctionEnd descending
  let $uu := doc("auctions.xml")//user/@username/string()
  return <p>
  {$p/productname}
  <info>
  {
    for $u in $uu
    let $f := doc("auctions")//bidproduct[@id=$p/@id]/bid[@user=$u]
    where $u != "Bart"

    return if(count($f) = 0) then
      <hb w="{ $u }">0</hb>
    else
      <hb w="{ $u }">1</hb>
  }
  </info></p>
}
</actn>
```

Geben Sie nun die Ausgabe von `auctions.xq` angewandt auf `auctions.xml` an.

Sie müssen sich nicht um Whitespaces kümmern.

```
<actn>
<p>
  <productname>Web Engineering</productname>
  <info>
    <hb w="Lisa">1</hb>
    <hb w="Homer">0</hb>
  </info>
</p>
<p>
  <productname>The Great Gatsby (Movie + Soundtrack)</productname>
  <info>
    <hb w="Lisa">0</hb>
    <hb w="Homer">1</hb>
  </info>
</p>
<p>
  <productname>XML in a Nutshell</productname>
  <info>
    <hb w="Lisa">0</hb>
    <hb w="Homer">0</hb>
  </info>
</p>
</actn>
```

Betrachten Sie das folgende XSLT Dokument **auctions.xsl**:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="2.0">
<xsl:output method="text"/>

<xsl:template match="auctions">
  <xsl:apply-templates select="//bidproduct"/>
</xsl:template>

<xsl:template match="bidproduct">
  <xsl:value-of select="//product[@id=current()/@id]/productname"/>
  ( <xsl:value-of select="//category[@id = //product[@id=current()/@id]/category/@ref][1]" /> )
  <xsl:for-each select="bid">
    <xsl:sort select="current()" order="descending"/>
    <xsl:apply-templates select="."/>
  </xsl:for-each>
</xsl:template>

<xsl:template match="bid">
  <xsl:variable name="userid" select="current()/@user" />
  $ <xsl:value-of select="."/> by <xsl:apply-templates select="//user[@username = $userid]"/>
</xsl:template>

<xsl:template match="user">
  <xsl:choose>
    <xsl:when test="fullname">
      <xsl:value-of select="fullname"/>
    </xsl:when>
    <xsl:otherwise>
      <xsl:value-of select="lastname"/> <xsl:value-of select="firstname"/>
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>

</xsl:stylesheet>
```

Geben Sie nun die Ausgabe von **auctions.xsl** angewandt auf **auctions.xml** an.

Sie müssen sich nicht um Whitespaces kümmern.

Web Engineering

(book)

\$ 450 by Bart Simpson

\$ 410 by Simpson Lisa

\$ 400 by Bart Simpson

The Great Gatsby (Movie + Soundtrack)

(movie)

\$ 300 by Bart Simpson

\$ 200 by Homer Simpson

XML in a Nutshell

(book)

\$ 150 by Bart Simpson

Total points: 75


```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE auctions SYSTEM "auctions.dtd">
<auctions>
  <products>
    <product id="p1" auctionEnd="2019-07-10">
      <productname>Web Engineering</productname>
      <description>This is the description of <a href="about:blank">Web Engineering</a></description>
      <category ref="cat1"/>
    </product>
    <product id="p2" auctionEnd="2019-06-25">
      <productname>The Great Gatsby (Movie + Soundtrack)</productname>
      <category ref="cat2"/>
      <category ref="cat3"/>
    </product>
    <product id="p3" auctionEnd="2016-06-30">
      <productname>XML in a Nutshell</productname>
      <description>XML in a Nutshell is a great book about XML.</description>
      <category ref="cat1"/>
    </product>
  </products>
  <categories>
    <category id="cat1">book</category>
    <category id="cat2">movie</category>
    <category id="cat3">music</category>
  </categories>
  <users>
    <user username="Bart"><fullname>Bart Simpson</fullname></user>
    <user username="Lisa"><firstname>Lisa</firstname><lastname>Simpson</lastname>
    </user>
    <user username="Homer"><fullname>Homer Simpson</fullname></user>
  </users>
  <bids>
    <bidproduct id="p1">
      <bid user="Bart">400</bid>
      <bid user="Lisa">410</bid>
      <bid user="Bart">450</bid>
    </bidproduct>
    <bidproduct id="p2">
      <bid user="Homer">200</bid>
      <bid user="Bart">300</bid>
    </bidproduct>
    <bidproduct id="p3">
      <bid user="Bart">150</bid>
    </bidproduct>
  </bids>
</auctions>
```