Klausur "Semistructured Data and XML"
Summer Term 2022
Prof. Dr. Wolfgang May
23. August 2022, 11:00–13:00
Working Time: 120 Minutes
(carried out as a computer-based ILIAS exam)

Vorname:

Nachname:

Matrikelnummer:

Setting: The usage of saxon (with the aliases saxonValid, saxonXQ, and saxonXSL defined as in the course) and xmllint (validation error messages are better with xmllint) is recommended. Web access (e.g. for XPath/XQuery and XSLT documentation) is allowed. It was also recommended to have the slides, and a condensed self-prepared "cheat sheet" (preparation of a cheat sheet is a very effective way to work through the materials).
Answers might be given in English or German (most answers are program code anyway).
In the text, the german translation is sometimes given in parentheses.
Give *all* answers via the ILIAS system.
Like in a "paper exam", also solutions that do maybe not work (or do not work completely) can be delivered and will be graded with appropriate partial points.
For **passing** the exam, **50** points are sufficient.

	Max. Punkte	Schätzung für "4"
Aufgabe 1 (XML)	20	15
Aufgabe 2 (DTD)	15	10
Aufgabe 3 (XPath (a))	3	3
Aufgabe 4 (XPath (b))	4	3
Aufgabe 5 (XPath/XQuery (c))	4	3
Aufgabe 6 (XPath/XQuery (d))	6	4
Aufgabe 7 (XPath/XQuery (e))	6	2
Aufgabe 8 (XPath/XQuery (f))	8	2
Aufgabe 9 (XPath/XQuery (g))	8	4
Aufgabe 10 (Data Integration and XPath/XQuery)	10	2
Aufgabe 11 (XSLT)	12	4
Aufgabe 12 (Miscellaneous)	4	3
Summe	100	55

Note:

Project: A Social Network Database for Dining and Restaurant Critics

The scenario is about a social-networks-style database on restaurants and ratings of their customer.

1. There are cities. They have a name and they are located in a country (we use just the same city names and country codes as in Mondial, this will be used in one of the exercises).

2. Every restaurant has a name and is located in a city. There may be restaurants with the same name in different cities ("La Trattoria", "Pizzeria Roma"), but not in a single city.

3. Every restaurant is classified according to its price category (extremely expensive=5, cheap=1).

4. For every restaurant, it is stored which kinds of food are served there (e.g., italian, fish, ...). A restaurant can feature several such styles of kitchen (e.g., japanese and fish fit well together). "Local" means that local food/specialities are served.

5. The following restaurants are located in Munich (located in Germany, "D"), listed with their price category and kinds of food:

   - Il Grappolo (category 3): italian, fish.
   - Makassar (3): indian.
   - Shoya (3): japanese, fish.
   - Augustiner (2): local.

6. The following restaurants are located in Lisbon (in Portugal, "P"):

   - Sete Mares (3): fish.
   - Bica do Sapata (4): portuguese, local.
   - Casanostra (3): italian.
   - Os Tibetanos (3): vegetarian.

7. The following restaurant is located in Almada (in Portugal, "P"):

   - Atira te ao Rio (4): portuguese, local.

8. The following restaurants are located in Rome (in Italy, "I"):

   - Agata e Romeo (4): italian, fish.
   - Le Relais de Jardin (5): french.
   - Margutta Vegetariano (4): vegetarian.

9. Furthermore, there are persons:

   - Every person has a username (without whitespaces) and lives in a city.

- Every person prefers one or more kinds of food, according to those listed for the restaurants ("local" means that the person is interested in the "local" food category wherever the person travels):
- *Alice* lives in Rome (in Italy, "I"), and she likes only vegetarian food.
- *Bob* lives in Munich, and he likes indian and local food.
- *Cristina* lives in Lisbon, and she likes italian, portuguese, and local food.

Thus, the database can be used for querying e.g., when Alice, Bob, and Cristina want to go out in the evening together for finding a restaurant that serves at least one favorite kind of kitchen for each of them.

Such groups can enter their dining appointments (restaurant, date) into the system, and also add the ratings (1...5, 5 is best; optionally also a short text) given by *each participant* afterwards:

- *Alice, Bob, and Cristina* had dinner in the *Augustiner* in Munich on July 25th, 2022. Alice rated it with 1 (there was a red-green caterpillar in her salad), Bob rated it with 5, and Cristina with 4.
- *Alice and Cristina* had dinner in the *Makassar* in Munich on July 27th, 2022. Alice rated it with 5, and Cristina with 4.
- *Alice and Cristina* had dinner in the *Os Tibetanos* in Lisbon on April 15th, 2022. Each of them rated it with 5.
- *Bob and Cristina* had dinner in the *Os Tibetanos* in Lisbon on June 30th, 2022. Bob rated it with 3, Cristina rated it with 4.
- *Alice and Bob* had dinner in the *Atira te ao Rio* in Almada on July 1st, 2022. Alice rated it with 3, Bob rated it with 4.
- *Alice and Bob* plan to have dinner in the *Relais de Jardin* in Rome on August 30th, 2022.

Exercise 1 (XML [20 Points])
Design an XML structure (use the frame given in file `exam.xml`) and fill it with some sample data (e.g. with some of the example data given in the text).

Copy-and-paste the XML from the file `exam.xml` afterwards (at the end of the exam, because it will be extended in later exercises) here:

Lösung

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE restaurants SYSTEM "exam.dtd">
<restaurants>
 <city name="Munich" country="D">
  <restaurant name="Il Grappolo" id="Grappolo-Munich" category="3">
   <style>italian</style>
   <style>fish</style>
  </restaurant>
  <restaurant name="Makassar" id="Makassar-Munich" category="3">
   <style>indian</style>
  </restaurant>
  <restaurant name="Shoya" id="Shoya-Munich" category="3">
   <style>japanese</style>
   <style>fish</style>
  </restaurant>
  <restaurant name="Augustiner" id="Augustiner-Munich" category="2">
   <style>local</style>
  </restaurant>
 </city>
 <city name="Lisbon" country="P">
  <restaurant name="Sete Mares" id="Mares-Lisbon" category="3">
   <style>fish</style>
  </restaurant>
  <restaurant name="Bica do Sapata" id="Bica-Lisbon" category="3">
   <style>portuguese</style>
   <style>local</style>
  </restaurant>
  <restaurant name="Casanostra" id="Casa-Lisbon" category="3">
   <style>italian</style>
  </restaurant>
  <restaurant name="Os Tibetanos" id="Tibet-Lisbon" category="3">
   <style>vegetarian</style>
  </restaurant>
 </city>
 <city name="Almada" country="P">
  <restaurant name="Atira te ao Rio" id="Rio-Almada" category="4">
   <style>local</style>
   <style>portuguese</style>
  </restaurant>
 </city>
 <city name="Rome" country="I">
  <restaurant name="Agata e Romeo" id="AgataR-Rome" category="3">
   <style>italian</style>
   <style>fish</style>
  </restaurant>
  <restaurant name="Le Relais de Jardin" id="Jardin-Rome" category="3">
   <style>french</style>
  </restaurant>
  <restaurant name="Margutta Vegetariano" id="Margutta-Rome" category="3">
   <style>vegetarian</style>
  </restaurant>
 </city>
 <person name="Alice" city="Rome">
  <likes>vegetarian</likes>
 </person>
 <person name="Bob" city="Munich">
  <likes>indian</likes>
  <likes>local</likes>
 </person>
 <person name="Cristina" city="Lisbon">
  <likes>italian</likes>
  <likes>portuguese</likes>
  <likes>local</likes>
 </person>
 <dinner restaurant="Augustiner-Munich" date="2022-07-25">
  <participant name="Alice" rating="1">
    There was a green-red caterpillar in the salad</participant>
  <participant name="Bob" rating="4"/>
  <participant name="Cristina" rating="4"/>
 </dinner>
 <dinner restaurant="Makassar-Munich" date="2022-07-27">
  <participant name="Alice" rating="5"/>
  <participant name="Cristina" rating="4"/>
 </dinner>
 <dinner restaurant="Tibet-Lisbon" date="2022-04-15">
  <participant name="Alice" rating="5"/>
  <participant name="Cristina" rating="5"/>
 </dinner>
 <dinner restaurant="Tibet-Lisbon" date="2022-05-30">
  <participant name="Bob" rating="3"/>
  <participant name="Cristina" rating="4"/>
 </dinner>
 <dinner restaurant="Rio-Almada" date="2022-06-01">
  <participant name="Alice" rating="3"/>
  <participant name="Bob" rating="4"/>
 </dinner>
 <dinner restaurant="Jardin-Rome" date="2022-08-30">
  <participant name="Alice"/>
  <participant name="Bob"/>
 </dinner>
</restaurants>
```

- note: NMTOKENS for @style would make comparisons harder since always the whole string is compared.
- note: the ‹dinner› elements could also be placed inside their restaurant elements, avoiding the usage of ID/IDREF for restaurants.

Exercise 2 (DTD [15 Points])
Give the DTD for your document developed in Exercise 1, use the file `exam.dtd`.
Use one of the calls

```
xmllint -loaddtd -valid --noblanks -noout exam.xml
saxonValid.bat -s:exam.xml
```

for validating it (note that xmllint provides better error messages).
Copy-and-paste the DTD from the file `exam.dtd` afterwards here:

Lösung

```
<!ELEMENT restaurants (city*, person*, dinner*)>
<!ELEMENT city (restaurant*)>
<!ELEMENT restaurant (style+)>
<!ATTLIST city name ID #REQUIRED>
<!ATTLIST restaurant id ID #REQUIRED
                     name CDATA #REQUIRED
                     category (1|2|3|4|5) #REQUIRED>
<!ELEMENT style (#PCDATA)>
<!ELEMENT person (likes+)>
<!ATTLIST person name ID #REQUIRED
                 city IDREF #REQUIRED>
<!ELEMENT likes (#PCDATA)>
<!ELEMENT dinner (participant+)>
<!ATTLIST dinner restaurant IDREF #REQUIRED
                 date CDATA #REQUIRED>
<!ELEMENT participant (#PCDATA)>
<!ATTLIST participant name IDREF #REQUIRED
                 rating (1|2|3|4|5) #IMPLIED>
```

- CDATA instead of (12|3|4|5)| -1/2

Exercise 3 (XPath (a) [3 Points])
Use your `exam.xml` XML file as a basis for solving this and the following exercises.
None of the results should contain duplicates.

Give an XPath query or an XQuery query that returns the *names* of those restaurants that serve vegetarian food and received at least one rating of 4 or better.
Write the query string in the file `query1.xq` and call it with

```
saxonXQ.bat -s:exam.xml query1.xq
```

Copy-and-paste the query from query1.xq afterwards here:

Lösung

```
//restaurant[style='vegetarian'
   and @id=//dinner[participant/@rating >= 4]/@restaurant]/@name/string()
```

Exercise 4 (XPath (b) [4 Points])
Give an XPath query or an XQuery query that returns the *names* of those cities where *no* restaurant serving vegetarian meals is stored in the database.
Write the XPath query string in the file `query2.xq` and call it with

```
saxonXQ.bat -s:exam.xml query2.xq
```

Copy-and-paste the query from query2.xq afterwards here:

Lösung

```
//city[not (restaurant[style="vegetarian"])]/@name/string()
```

Exercise 5 (XPath/XQuery (c) [4 Points])
Give an XPath or XQuery query that yields the names of all restaurants where Bob and Cristina can meet for dinner, and both find an offer that they like.
Copy-and-paste the query from query3.xq afterwards here:

Lösung

```
  //restaurant[ style = //person[@name='Bob']/likes
          and   style = //person[@name='Cristina']/likes]
  /@name/string()
```

Exercise 6 (XPath/XQuery (d) [6 Points])
Give an XQuery query which, for every restaurant, yields the average of its ratings. The results should be ordered by that average descending in the form

```
<restaurant name="..." city="..." avg="..."/>
```

Copy-and-paste the query from query4.xq afterwards here:

Lösung

```
for $r in //restaurant[//dinner/@restaurant=./@id]
let $avg := avg(//dinner[@restaurant=$r/@id]/participant/@rating)
where $avg  (: means: is not null, otherwise returns NaN - not a number :)
order by $avg descending
return
<restaurant name="{string($r/@name)}"
            city="{string($r/../@name)}" avg ="{$avg}"/>
```

Exercise 7 (XPath/XQuery (e) [6 Points])
Give an XQuery query that yields all pairs (Person, City) such that every kind of food that this person likes is offered in at least one restaurant in this city. The results should be of the format

```
<result person="..." city="..."/>.
```

Copy-and-paste the query from query5.xq afterwards here:

Lösung

```
for $p in //person, $c in //city
where every $st in $p/likes
satisfies $st = $c/restaurant/style
return <result person="{$p/@name}" city="{$c/@name}"/>
```

Exercise 8 (XPath/XQuery (f) [8 Points])
Give an XQuery query that, for each person P, returns a list (without duplicates) of all cities where this person has rated some restaurant. For each person, the result should be of the form

```
<result person="name-of-P">
   name-of-city-C1 ... name-of-city-Cn
</result>
```

(no duplicate city names, in any order, with arbitrary whitespaces)
Copy-and-paste the query from query6.xq afterwards here:

Lösung

```
for $p in //person
return <result>
  { $p/@name,
    //dinner[participant[@name = $p/@name and @rating]]
      /id(@restaurant)/parent::city/@name/string()
  }
  </result>
```

- Note: the XPath expression automatically removes duplicates (after each step of its evaluation), so an explicit `distinct-values` is not necessary; otherwise the sample data would return "Munich" several times.
- another possibility to eliminate duplicate cities is to run the for-loop over the cities:

```
for $p in //person
return <result>
 { $p/@name,
   for $c in //city[. =
       //dinner[participant[@name = $p/@name and @rating]]
         /id(@restaurant)/parent::city]
   return $c/@name/string()
 }
 </result>
```

Exercise 9 (XPath/XQuery (g) [8 Points])
Give an XQuery query that yields for each city and each person the number of restaurants in that city that serve some style of food that this person likes. Only those cities should be given where at least two such restaurants are located.
For each pair, the result should be of the form

```
<result person="name-of-P" city="name-of-city"> number </result>
```

Copy-and-paste the query from query7.xq afterwards here:

Lösung

```
for $c in //city, $p in //person
let $restaurants := $c/restaurant[style=$p/likes]
where count($restaurants) >= 2
return <result person="{$p/@name}" city="{$c/@name}">
          {count($restaurants)}
       </result>
```

Exercise 10 (Data Integration and XPath/XQuery [10 Points])
This exercise combines the data in your exam.xml file with the data in mondial (use your local mondial.xml or the one at `http://www.dbis.informatik.uni-goettingen.de/Mondial/mondial-europe.xml`.

Assume here that the name of each used city is unique in its country, so one does not need to consider the province names.

Give an XQuery query (query-int.xq) that returns the names of the *rivers* where the cities are located at, where Alice gave a rating to some restaurant.

Copy-and-paste the query from query-int.xq afterwards here:

Lösung

```
distinct-values(
for $c in //city[restaurant[@id=
       //participant[@name='Alice' and @rating]/../@restaurant]]
let $mcity :=
  doc("http://www.dbis.informatik.uni-goettingen.de/Mondial/mondial.xml")
    //id($c/@country)//city[name=$c/@name]
return $mcity/located_at/id(@river)/name[1]
)
```

```
let $rivers :=
( for $c in //participant[@name='Alice' and @rating]
                         /../id(@restaurant)/parent::city
  let $mcity :=
    doc("http://www.dbis.informatik.uni-goettingen.de/Mondial/mondial.xml")
      //id($c/@country)//city[name=$c/@name]
  return $mcity/located_at/id(@river))
return distinct-values($rivers/name[1])
```

- The first solution is brute-force using `distinct-values`. Note that it avoids duplicate *cities* by iterating already over the results of an XPath expression wrt. exam.xml. But still, duplicate rivers in case that several cities are located at the same river must be removed afterwards (this is only possible with literals).
(otherwise duplicate output "Tejo", for Lisbon and Almada).

- The second solution is similar.

- A solution without `distinct-values`: exploiting that the XPath expression in the return clause eliminates duplicate rivers:

```
 let $cities := //participant[@name='Alice' and @rating]
                             /../id(@restaurant)/parent::city,
     $mcities :=
      for $c in $cities
      return doc("http://www.dbis.informatik.uni-goettingen.de/Mondial/mondial.xm
               //city[name=$c/@name and @country=$c/@country]
return $mcities/located_at/id(@river)/name[1]/string()
```

- Note: using the city name and the country code for matching requires to use XQuery with at least one variable for the cities. When only the city name is used, much simpler queries are possible.

- Note: in all cases, for navigating to the rivers, the id()-function is applied inside Mondial. This shows that the XQuery engine actually also loads the referenced DTD. (It would even distinguish ids between different documents, in case that exam.xml would also treat countries with their codes as IDs locally.)

Exercise 11 (XSLT [12 Points])
Extend the given XSL stylesheet frame `exam.xsl` to an XSLT stylesheet that returns for every person a simple HTML table that lists the person's name, and then all ratings (restaurant name, place, date, rating, optionally the rating's text) that the person made (output should be in chronological order).
For ratings with "1" and "2" the text color should be red, for "4" and "5" it should be green, and for "3" it should be simply black (coloring is done in HTML by `<font color="red">...</font>`)

Use the following call to execute it:

```
saxonXSL.bat -s:exam.xml exam.xsl
```

Copy-and-paste the XSLT stylesheet from exam.xsl afterwards here:

Lösung

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  version="2.0">

  <xsl:template match="restaurants">
    <html><body>
    <xsl:apply-templates select="person"/>
    </body></html>
  </xsl:template>

  <xsl:template match="person">
    <table>
    <tr><td colspan="5">
      <xsl:value-of select="@name"/>
    </td></tr>
    <xsl:apply-templates
       select="//participant[@name = current()/@name and @rating]">
      <xsl:sort select="../@date"/>
    </xsl:apply-templates>
    </table>
  </xsl:template>

  <xsl:template match="participant">
    <tr>
      <td><xsl:value-of select="id(../@restaurant)/@name"/></td>
      <td><xsl:value-of select="id(../@restaurant)/../@name"/></td>
      <td><xsl:value-of select="../@date"/></td>
      <td>
        <xsl:if test="@rating &lt; 3">  <!-- lt is escaped less-than -->
          <font color="red"><xsl:value-of select="@rating"/></font>
        </xsl:if>
        <xsl:if test="@rating > 3">
          <font color="green"><xsl:value-of select="@rating"/></font>
        </xsl:if>
        <xsl:if test="@rating = 3">
          <xsl:value-of select="@rating"/></xsl:if>
      </td>
      <td><xsl:value-of select="text()"/></td>
    </tr>
  </xsl:template>

</xsl:stylesheet>
```

Exercise 12 (Miscellaneous [4 Points])
Put the following data models into the correct temporal order (oldest first):

- RDF
- Relational Data Model
- OEM/Tsimmis

- Network Data Model/CODASYL
- F-Logic
- ODMG/OIF
- XML

Lösung

1. Network Data Model: 1964, CODASYL standard 1971
2. Relational Data Model 1970, SEQUEL/SQL since 1975, first product (Oracle V2) 1979, formal SQL standard only later since 1986(!)
3. F-Logic: 1989
4. ODMG/OIF: ODMG 1.0 1993
5. OEM/Tsimmis: 1995
6. XML: development started 1996, first official proposal in February 1998
7. RDF: proposal 1997, standardized 1999

- important to know: netw-rel-{F-Logic/OEM/ODMG}-XML-RDF: yields 3P
- 3.5P if ODMG listed before{F-Logic|OEM} because intuitively, ODMG is less progressive than F-Logic and OEM (reasonable answer for participants who did not look up the exact years in the slides)

The following frames can be used:

- Frame for XML file `exam.xml`:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE put-name-here SYSTEM "exam.dtd">
  to be extended here
```

- Frame for XML stylesheet `exam.xsl`:

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  version="2.0">
  to be extended here
</xsl:stylesheet>
```