Klausur "Semistructured Data and XML"
Summer Term 2022
Prof. Dr. Wolfgang May
23. August 2021, 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	Erreichte Punkte
Aufgabe 1 (XML)	20	
Aufgabe 2 (DTD)	15	
Aufgabe 3 (XPath (a))	3	
Aufgabe 4 (XPath (b))	4	
Aufgabe 5 (XPath/XQuery (c))	4	
Aufgabe 6 (XPath/XQuery (d))	6	
Aufgabe 7 (XPath/XQuery (e))	6	
Aufgabe 8 (XPath/XQuery (f))	8	
Aufgabe 9 (XPath/XQuery (g))	8	
Aufgabe 10 (Data Integration and XPath/XQuery)	10	
Aufgabe 11 (XSLT)	12	
Aufgabe 12 (Miscellaneous)	4	
Summe	100	

Note:

Name: MatNr.:

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:

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:

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:

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:

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:

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:

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:

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:

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:

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:

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:

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

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>
```