2023-12-13 - M-CS - Data Science - 202300200 - Topic SEMI

Course: M-CS Data Science 202300200 – 2023-12-13 1 to 10

This test is for the Topic SEMI. You may use your (not graphing) calculator. The length of the test is 1 hour, but to be able to finish both tests, you have from 8:45 – 11:45.

Number of questions: 6

You can score a total of 100 points for this exam, you need 55 points to pass the exam.

1 What does the "well-formedness" property mean for an XML document?

5 pt.

a. The XML document adheres to a given schema.

b. The start and end tags are properly deliminated between '<' and '>' symbols.

c. Start and end tags are matched precisely and are properly nested.

d. The XML document can be directly one-to-one converted to a JSON object.

2 RDF is a graph-based data model; what do the nodes and the edges in this graph represent?

5 pt.

a. Nodes are XML elements, and edges represent the hierarchical structure between XML elements.

b. Nodes are entities, and edges represent properties or relationships with other entities.

c. Nodes are entities, and edges represent their data types.

d. Nodes are XML elements, and edges represent their attributes.

3 In the story about the history of XML, the "General Markup Language" (GML) introduced in 1969 was discussed. It allowed "heading" to be used instead of ".bf.ce.roman16" in a document such as below. XML evolved from GML. How can we see this capability back in XML?

5 pt.

```
.roman16
.ce
.bf
Introduction
.roman12
.ju
.ll 39
.ss
The work of authors,
publishers, and
researchers involves,
in varying degrees,
three recognizable
```

a. "heading" can appear as element name in an XML document with an associated declaration in a style sheet that specifies how heading elements are to be displayed.

b. <bf><ce><roman16>heading</roman16></ce><bf> can appear as in an XML document in this way specifying how headings are to be displayed.

c. <heading style="bf;ce;roman16"> can appear as in an XML document in this way specifying how headings are to be displayed.

d. heading=".bf.ce.roman16" can appear as attribute in an XML document in this way specifying how headings are to be displayed.

4 In what way(s) is/are JSON and XML the same?
5 pt. (multiple answers possible)

- **a.** They both represent tree-structured data.
- **b.** Both JSON and XML are human-readable and machine-readable facilitating easy understanding and parsing of data.
- **c.** They both represent graph-structured data.
- **d.** Both JSON and XML are data interchange formats used to represent and exchange structured data.
- **e.** JSON is a newer and more advanced version of XML, so they are largely the same with only a few extra capabilities.
- **f.** JSON and XML serve the same purpose and are directly interchangeable in any data exchange scenario.
- **g.** JSON is exclusively used for client-side web development, while XML is used for server-side data storage and exchange, but data-wise they are the same.

5 Please find below a small XML document containing data on music albums.

```
<catalog>
  <!-- Some albums before ... -->
  <album>
    <title year="1973">The Dark Side of the Moon</title>
    <artist>Pink Floyd</artist>
    <genre>Progressive Rock</genre>
    <tracks>
      <track>Speak to Me</track>
      <track>Breathe</track>
      <track>On the Run</track>
      <!-- Additional tracks for the album ... -->
    </tracks>
  </album>
  <!-- Some albums after ... -->
</catalog>
```

Complete the following XPath query which determines the years of albums from artist "Pink Floyd". The query first finds all 'artist' elements with text "Pink Floyd", then navigates to the title-elements of these albums, and from there navigates to the year.

`//artist[. = "Pink Floyd"]/`

(a) **A.** `parent` **B.** `following-sibling` **C.** `ancestor` **D.** `preceding` **E.** `following` **F.** `preceding-sibling` **G.** `child` **H.** `descendant` (6 pt.)

`:: title` **(b)** **A.** `/@year` **B.** `/child::year` **C.** `/year` **D.** `/value::year` (6 pt.)

The following XQuery provides an HTML-based overview of all albums of artist "Pink Floyd" with the number of tracks on each album. The XML document is assumed to be stored under the name "catalog.xml".

```
<html>
<h1>Overview Pink Floyd albums</h1>
<ul>{
  for $album in doc("catalog.xml")//album
  let $title := $album/title/text()
  where $album/artist = "Pink Floyd"
  return
    <li>{$title} ({count($album//track)} tracks)</li>
}</ul>
</html>
```

7 pt. **c.** What would happen if we were to forget all curly brackets '{' and '}'?

a. Nothing special; the query still works properly. The curly brackets are optional and can be used to indicate where the XQuery begins and ends.

b. The query would run, but would produce a HTML-page with the query text in it instead of a list of albums.

c. The query gives an error, because if it is not a proper XQuery anymore.

7 pt. **d.** What happens when there is an album that has no tracks and why?

a. It will properly say "(0 tracks)", because "$album//track" evaluates to the empty sequence and the count of an empy sequence is 0.

b. It gives an error, because "$album//track" cannot be evaluated. One should include a test for an empty track list in the query.

c. It evaluates to NULL, because "$album//track" evaluates to NULL, and the count of a NULL-value is NULL.

d. It evaluates to 1, because "$album//track" evaluates to the string "NULL", and the count of a single string value is 1.

7 pt. **e.** What does the "/text()" exactly do in the line "let $title := $album/title/text()"

a. It navigates to the child text node of the title-element.

b. It runs the 'text' function which strips the title-element of any non-text items such as attributes.

c. It runs the 'text' function which determines the 'string value' of the element, which includes the string values of the attributes, so in this case it actually evaluates to "1973 The Dark Side of the Moon"

d. It runs the 'text' function which returns the text contents of the title-element.

7 pt. **f.** Could the document "catalog.xml" be produced with SQL/XML from data in a relational database?

- **a.** Yes, the structure of the algorithms is quite regular and the data probably resided in two tables ("albums" and "tracks"). The "catalog.xml" document can be produced with SQL/XML using constructor functions XMLELEMENT, XMLATTRIBUTES, and XMLFOREST.
- **b.** No, although the structure of the algorithms is quite regular, the data probably resided in two tables ("albums" and "tracks") and SQL/XML can only produce XML from data that is stored in one table.
- **c.** No, the document "catalog.xml" is document-centric XML and too irregular to be produced with SQL/XML from data in a relational database.
- **d.** Yes, the structure of the algorithms is quite regular and the data probably resided in two tables ("albums" and "tracks"). The "catalog.xml" document can be produced with SQL/XML using constructor functions XMLELEMENT, XMLATTRIBUTES, and XMLAGG.

6 Please find below a small RDF document containing data on cars.

```
<?xml version="1.0" encoding="UTF-8"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
         xmlns:ex="http://example.org/cars#">

<!-- Car Resources -->
<rdf:Description rdf:about="http://example.org/cars/car1">
  <rdf:type rdf:resource="http://example.org/cars#Car"/>
  <ex:manufacturer>Toyota</ex:manufacturer>
  <ex:model>Camry</ex:model>
  <ex:year>2022</ex:year>
</rdf:Description>

<rdf:Description rdf:about="http://example.org/cars/car2">
  <rdf:type rdf:resource="http://example.org/cars#Car"/>
  <ex:manufacturer>Ford</ex:manufacturer>
  <ex:model>Mustang</ex:model>
  <ex:year>2021</ex:year>
</rdf:Description>

<!-- Property Definitions -->
<rdf:Description rdf:about="http://example.org/cars#Car">
  <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
  <rdfs:label xml:lang="en">Car</rdfs:label>
</rdf:Description>

<rdf:Description rdf:about="http://example.org/cars#manufacturer">
  <rdf:type rdf:resource="http://www.w3.org/1999/02/
22-rdf-syntax-ns#Property"/>
  <rdfs:label xml:lang="en">Manufacturer</rdfs:label>
</rdf:Description>

<rdf:Description rdf:about="http://example.org/cars#model">
  <rdf:type rdf:resource="http://www.w3.org/1999/02/
22-rdf-syntax-ns#Property"/>
  <rdfs:label xml:lang="en">Model</rdfs:label>
</rdf:Description>

<rdf:Description rdf:about="http://example.org/cars#year">
  <rdf:type rdf:resource="http://www.w3.org/1999/02/
22-rdf-syntax-ns#Property"/>
  <rdfs:label xml:lang="en">Year</rdfs:label>
</rdf:Description>

</rdf:RDF>
```

Given also an example SPARQL query to be run on this RDF document:

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX ex: <http://example.org/cars#>

SELECT ?car ?model ?year
WHERE {
  ?car rdf:type ex:Car.
  ?car ex:manufacturer "Ford".
  ?car ex:model ?model.
  ?car ex:year ?year.
}
```

7 pt. **a.** What would happen if we would forget to include the 'PREFIX' lines?

- **a.** The query gives an error; every prefix needs to be declared, because otherwise SPARQL doesn't know how to determine the URI.
- **b.** They only define shorthands, but are actually not necessary here. The query will run fine without them and will produce the intended result.
- **c.** The query would run, but would produce an empty answer, because, for example, the URI "ex:manufacturer" in the query doesn't match the URI "http://example.org/cars#manufacturer" in the document.

6 pt. **b.** The RDF document looks like XML. Is it? If so, why?

- **a.** No, RDF is not XML. It looks very similar, but there are notable differences which, among others, make it that an RDF document cannot be parsed by an XML parser.
- **b.** Yes, the RDF document is XML. RDF is actually one of the many XML-standards formally established and maintained by the W3C.
- **c.** Yes, the RDF document is XML, but it is not a formal XML standard. RDF is developed by a community just like JSON and it is not guaranteed that it can be parsed for every XML parser.

6 pt. **c.** Which description most accurately describes what a SPARQL query represents?

- **a.** It represents a declarative program just like XQuery which is compiled and optimized and then executed.
- **b.** It represents a subgraph template with variables as placeholders, which is matched against the document providing values for these placeholders for every match.
- **c.** It represents a query just like an SQL query but for RDF data; it selects properties of RDF nodes for which certain conditions hold.

7 pt. **d.** RDF data is composed of "triples". Which of the following are triples defined in the document above?

- **a.** ("http://example.org/cars#Car", "http://www.w3.org/1999/02/22-rdf-syntax-ns#type" , "http://www.w3.org/1999/02/22-rdf-syntax-ns#Property")
- **b.** ("http://example.org/cars/car1" , "http://www.w3.org/2000/01/rdf-schema#label" , "Manufacturer")
- **c.** ("http://example.org/cars/car1" , "http://www.w3.org/1999/02/22-rdf-syntax-ns#type" , "http://example.org/cars#Car")
- **d.** ("http://example.org/cars#manufacturer" , "http://www.w3.org/2000/01/rdf-schema#label" , "Manufacturer")
- **e.** ("http://example.org/cars/car1" , "http://example.org/cars#manufacturer" , "Toyota")

7 pt. **e.** RDF data is composed of triples. Does a SPARQL query also contain triples?

- **a.** No, it does not.
- **b.** Yes, they are in the WHERE-clause and each triple is separated by a '.' symbol, e.g., "?car rdf:type ex:Car" is the first triple in the query.
- **c.** Yes, there can be only one and it is in the SELECT clause: "?car ?model ?year" is a triple.
- **d.** Yes, they are both in the WHERE-clause as well as the SELECT-clause: both "?car rdf:type ex:Car" and "?car ?model ?year" are examples of triples in the given SPARQL query.

7 pt. **f.** Can RDF data have a schema specifying its structure?

- **a.** Yes, a schema is called an "ontology" in the Linked Open Data and Semantic Web approach.
- **b.** No, RDF data is semi-structured and does not have a schema.
- **c.** Yes and No: the structure can be specified, but it is not in a separate document but is specified as part of the RDF data itself using the "rdf:type" properties, which refer to entities representing types.
- **d.** No, the structure of RDF data is "triples". No need to specify that with a schema, because it is the same for all RDF documents.

Name:			Signature:
Date:	/ /	Birth date:	/ /

Course: M-CS-202300200-1A M-CS Data Science 202300200 - Category: 2023-12-13 1 to 10 - Questions: 2023-12-13 - M-CS - Data Science - 202300200 - Topic SEMI - Code: 2324-1B-Example-Exam

This test is for the Topic SEMI. You may use your (not graphing) calculator. The length of the test is 1 hour, but to be able to finish both tests, you have from 8:45 – 11:45.

1 5 pt. A ◯ B ◯ C ◯ D ◯

2 5 pt. A ◯ B ◯ C ◯ D ◯

3 5 pt. A ◯ B ◯ C ◯ D ◯

4 5 pt. A ☐ B ☐ C ☐ D ☐ E ☐ F ☐ G ☐

There are multiple answers possible

5 40 pt.

a. A ◯ B ◯ C ◯ D ◯ E ◯ F ◯ G ◯ H ◯

b. A ◯ B ◯ C ◯ D ◯

c. A ◯ B ◯ C ◯

d. A ◯ B ◯ C ◯ D ◯

e. A ◯ B ◯ C ◯ D ◯

f. A ◯ B ◯ C ◯ D ◯

			A	B	C	D	E
6 40 pt.	**a.**		◯	◯	◯		
	b.		◯	◯	◯		
	c.		◯	◯	◯		
	d.		▢	▢	▢	▢	▢
	e.		◯	◯	◯	◯	
	f.		◯	◯	◯	◯	

There are multiple answers possible

Correction model

1. C
5 pt.

2. B
5 pt.

3. A
5 pt.

4. A
B
D
5 pt.

5.
40 pt.

a.		F
b.		A
c.	7 pt.	B
d.	7 pt.	A
e.	7 pt.	A
f.	7 pt.	D

5f: (disclaimer: from the top of my head, so could contain errors) Logical structure of the tables would be: albums (albumid, title, year, artist, genre) and tracks(trackid, albumid, name). Then an SQL/XML query to produce catalog.xml would be:

```
SELECT XMLELEMENT(NAME album,
  XMLELEMENT(NAME title, XMLATTRIBUTES(year),title),
  XMLELEMENT(NAME artist, artist),
  XMLELEMENT(NAME genre, genre),
  XMLELEMENT(NAME tracks,
    XMLAGG(XMLELEMENT(NAME track, name))
  )
)
FROM albums, tracks
WHERE albums.albumid = tracks.albumid
GROUP BY tracks.albumid
```

6.
40 pt.

a.	7 pt.	A
b.	6 pt.	B
c.	6 pt.	B
d.		A
		B
	1 pt.	C
	1 pt.	D
	1 pt.	E
	Bonus: 0 pt.	
e.	7 pt.	B
f.		A

6d: This means that C, D, and E are correct, and A and B are incorrect.

Caesura

Points scored	Grade
100	10
99	9.9
98	9.8
97	9.7
96	9.6
95	9.5
94	9.4
93	9.3
92	9.2
91	9.1
90	9.0
89	8.9
88	8.8
87	8.7
86	8.6
85	8.5
84	8.4
83	8.3
82	8.2
81	8.1
80	8.0
79	7.9
78	7.8
77	7.7
76	7.6
75	7.5
74	7.4
73	7.3
72	7.2
71	7.1
70	7.0
69	6.9
68	6.8
67	6.7
66	6.6
65	6.5
64	6.4
63	6.3
62	6.2
61	6.1
60	6.0
59	5.9
58	5.8
57	5.7
56	5.6
55	5.5
54	5.4
53	5.3
52	5.3
51	5.2
50	5.1
49	5.0
48	4.9
47	4.8
46	4.8
45	4.7
44	4.6
43	4.5
42	4.4
41	4.4
40	4.3

39	4.2
38	4.1
37	4.0
36	3.9
35	3.9
34	3.8
33	3.7
32	3.6
31	3.5
30	3.5
29	3.4
28	3.3
27	3.2
26	3.1
25	3.0
24	3.0
23	2.9
22	2.8
21	2.7
20	2.6
19	2.6
18	2.5
17	2.4
16	2.3
15	2.2
14	2.1
13	2.1
12	2.0
11	1.9
10	1.8
9	1.7
8	1.7
7	1.6
6	1.5
5	1.4
4	1.3
3	1.2
2	1.2
1	1.1
0	1.0