Exam - Solution

SDXML
Models and languages for handling semi-structured data and XML

YYYY-MM-DD

- The exam consists of three parts:
  Part 1 Theory and modeling (questions 1, 2, 3)
  Part 2 Query languages for SSD and XML (questions 4, 5)
  Part 3 XML and relational databases (question 6)

- The exam is graded based on the grading scale F/Fx/E/D/C/B/A.

- In order to pass the exam, the student must perform at a certain level on each part and at a certain level in total, thus fulfilling all three course goals (which correspond to the three exam parts). The exact levels for each grade are:

	Whole exam	Per part
Minimums for A	92%	84%
Minimums for B	84%	73%
Minimums for C	76%	62%
Minimums for D	68%	51%
Minimums for E	60%	40%

  In the unlikely event that a student achieves a very uneven result (very good result, 85% or more, in one or two parts, without reaching the minimums for E), the student will be offered the option of skipping parts during the retake. This will be denoted by the grade Fx and will be explained in the feedback to the exam. Any student that chooses to use this option will be able to receive at most the grade E. Students that have been offered this option can ignore it. When this option is used, the skipped parts are excluded from the grade calculation and the minimums for E apply to the remaining parts.

- Specify the **course code**, the **date**, the **exercise number** and your **anonymous examinee code** on each submitted page!

- Fill out the scanning page with the **number of submitted pages** and specify **which questions that have been answered**!

- No aids allowed.

Reference

SQL

Syntax for SQL SELECT:

```
SELECT [DISTINCT] <column list>
FROM <table list>
[WHERE <conditions>]
[GROUP BY <column list>
  [HAVING <conditions>]]
[ORDER BY <column list>]
```

Common SQL/XML functions:
XMLELEMENT, XMLATTRIBUTES, XMLFOREST, XMLAGG, XMLCONCAT, XMLEXISTS, XMLQUERY, XMLTABLE, XMLCOMMENT, XMLPI, XMLCAST

XML Schema

Common elements:
schema, element, attribute, complexType, simpleType, group, all, sequence, choice, restriction, extension, simpleContent, complexContent

XQuery

Syntax for XQuery FLWOR:

```
for <loop variables>
let <variable assignments>
where <conditions>
(group by <grouping expressions>) only in XQuery 3
order by <sorting expressions>
return <result>
```

Common XQuery functions:
distinct-values(), count(), sum(), min(), max(), avg(), empty(), exists(), not(), data()

XSLT

Common elements:
transform, output, variable, template, for-each, for-each-group, apply-templates, call-template, if, choose, when, otherwise, element, attribute, comment, processing-instruction, value-of, text, copy, copy-of, sort, param, with-param

SQL Server SQL

SQL clause: FOR XML PATH | AUTO | RAW
Relevant keywords: ELEMENTS, ROOT, TYPE
XML methods: query, value, nodes, exist, modify
XQuery functions: sql:column, sql:variable

Question 1 (6 points)

Question 1 consists of 6 terms to be explained/defined in short. The answers only need to illustrate understanding. No need for any long essays.

Explain the following terms:

1. Valid XML
2. Infoset
3. processing-instruction
4. JSON Schema
5. shredding
6. SVG

1. Valid XML means that an XML document conforms to the rules in the associated DTD or XML Schema. Valid XML is always will-formed.
2. An abstract representation of XML documents. Infoset represents only significant information. Infoset cannot represent XML fragments and does not care about the validity of the XML document or the associated DTD or XML Schema, which includes data types.
3. A special type of XML node that has the following format: <?name content?>. The XML declaration is a processing-instruction. Processing-instruction (as the name implies) are used to provide information to the XML processor (how to handle or render XML).
4. JSON Schema is used to specify rules for JSON objects. JSON Schema uses JSON syntax, which means that a JSON Schema is JSON.
5. Shredding means breaking up XML into its nodes and data. Most commonly in order to put the data in a relational database.
6. Scalable Vector Graphics: An XML-based language for representing vector graphics. SVG defines elements and attributes for describing circles, rectangles, ellipses, lines and text, and properties for their coordinates, dimensions, colors, transparency, font, etc.

Question 2 (2 points)

Create a JSON object that conforms to following JSON Schema!

```
{
  "$schema": "https://json-schema.org/draft/2020-12/schema",
  "$id": "http://dsv.su.se/SDXML/jsonschema/song",
  "type": "object",
  "title": "A song",
  "description": "An object representation of a song",
  "properties": {
    "name": {"type": "string"},
    "lyrics": {"type": "string"},
    "year": {"type": "integer"},
    "writers": {
      "type": "array",
      "items": {
        "type": "object",
        "properties": {
          "name": {"type": "string"},
          "yearofbirth": {"type": "integer"}
        },
```

```
            "additionalProperties": false,
            "required": ["name", "yearofbirth"]
          },
          "minItems": 1
        },
        "knownPerformers": {"type": "array", "items": {"type": "string"}}
    },
    "additionalProperties": false,
    "required": ["name", "year", "writers"]
}
```

A minimal valid JSON could be the following:

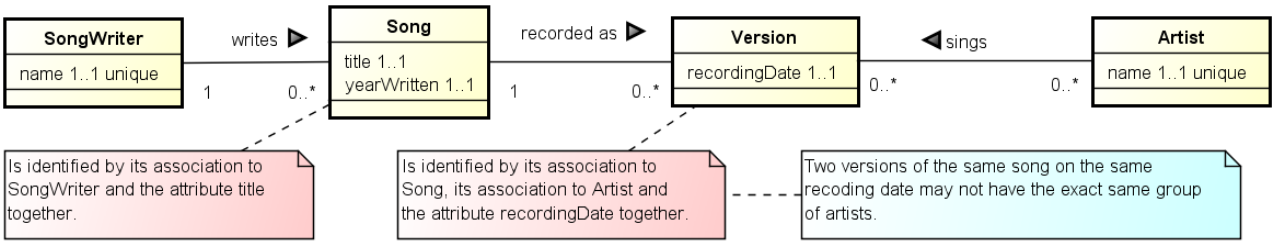
```
{"name" : "Hello",
 "year" : 1999,
 "writers" : [{"name" : "John", "yearofbirth" : 1980}]}
```

If we include the allowed optional properties, we could get something a little bigger:

```
{"name" : "Hello",
 "lyrics" : "Hello, Hello, Hey, Hey",
 "year" : 1999,
 "writers" : [{"name" : "John", "yearofbirth" : 1980}, {"name" : "Marie", "yearofbirth" : 1981}],
 "knownPerformers" : ["James", "Will", "Jenna"]
}
```

Question 3 (3 points)

Construct an XML document (with sample data) and a corresponding XML Schema for representing the same information as the provided conceptual model! No need to explicitly define what is unique. The solution must consider integrity rules and avoid unnecessary redundancy.



First, we have an XML document. By nesting elements, we can force things to belong together and avoid having unnecessary references. For example, a song can only exist inside the element of the song writer, thus no song can be without song writer and no song can refer to a non-existing song writer (if we had a reference). Similarly, a version belongs to a song. Since artists need to be reused for many versions and also exist even without versions, they are place directly under the root element with references from the versions. We could refer to the versions from the artists, but that would be more complicated.

```
<?xml version="1.0" encoding="UTF-8"?>
<Data>
    <SongWriter name="Jimmy Thörnfeldt"/>
    <SongWriter name="Linnea Deb">
        <Song title="Every Second" yearWritten="2019">
            <Version recordingDate="2019-10-10">
                <SingingArtist id="1"/>
                <SingingArtist id="2"/>
            </Version>
            <Version recordingDate="2019-12-12">
                <SingingArtist id="3"/>
            </Version>
            <Version recordingDate="2019-12-12">
                <SingingArtist id="2"/>
            </Version>
            <Version recordingDate="2020-05-05"/>
            <!-- More Version elements with zero or more SingingArtist -->
            <!-- No identical Version elements within the same Song -->
        </Song>
        <Song title="Voices" yearWritten="2021"/>
        <!-- More Song elements with zero or more Version -->
    </SongWriter>
    <!-- More SongWriter with zero or more Song -->
    <Artist id="1" name="Eric Saade"/>
    <Artist id="2" name="Klara Hammarström"/>
    <Artist id="3" name="Malou Prytz"/>
    <Artist id="4" name="Lisa Ajax"/>
    <!-- More Artist elements -->
</Data>
```

The following XML Schema defines the structure of the XML document. Only the basic rules of multiplicities and data types are defined.

```
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema">
    <element name="Data">
        <complexType>
            <sequence>
                <element name="SongWriter" type="SongWriterCT" minOccurs="0" maxOccurs="unbounded"/>
                <element name="Artist" minOccurs="0" maxOccurs="unbounded">
                    <complexType>
                        <attribute name="id" type="integer" use="required"/>
                        <attribute name="name" type="string" use="required"/>
                    </complexType>
                </element>
            </sequence>
        </complexType>
    </element>
```

```
    <complexType name="SongWriterCT">
        <sequence>
            <element name="Song" minOccurs="0" maxOccurs="unbounded">
                <complexType>
                    <sequence>
                        <element name="Version" type="VersionCT" minOccurs="0" maxOccurs="unbounded"/>
                    </sequence>
                    <attribute name="title" type="string" use="required"/>
                    <attribute name="yearWritten" type="integer" use="required"/>
                </complexType>
            </element>
        </sequence>
        <attribute name="name" type="string" use="required"/>
    </complexType>
    <complexType name="VersionCT">
        <sequence>
            <element name="SingingArtist" minOccurs="0" maxOccurs="unbounded">
                <complexType>
                    <attribute name="id" type="integer" use="required"/>
                </complexType>
            </element>
        </sequence>
        <attribute name="recordingDate" type="date" use="required"/>
    </complexType>
</schema>
```

Question 4 (2 + 2 + 2 = 6 points)

Consider the XML document after question 5 that only contains sample data in order to illustrate the structure.

a) Write XQuery for the following:
Retrieve information about every member of parliament according to the following structure:

```
<Result>
    <MP Name="" Party="" NumberOfCommittees=""/>
    <MP Name="" Party="" NumberOfCommittees=""/>
    <MP Name="" Party="" NumberOfCommittees=""/>
</Result>
```

```
element Result {
 for $mp in //MP
 return element MP {attribute Name {$mp/@name},
                    attribute Party {$mp/../@name},
                    attribute NumberOfCommittees {count($mp/Assignment)}}
}
```

b) Write XQuery for the following:
Retrieve information about each committee according to the following structure!

```
<Result>
    <Committee Name="">
        <Party Name="" NumberOfMembers=""/>
        <Party Name="" NumberOfMembers=""/>
        <Party Name="" NumberOfMembers=""/>
    </Committee>
    <Committee Name="">
        <Party Name="" NumberOfMembers=""/>
        <Party Name="" NumberOfMembers=""/>
    </Committee>
</Result>
```

Note: NumberOfMembers shall be the number of MPs of the party in the committee independent of their role.

```
element Result {
 for $c in distinct-values(//@committee)
 return element Committee {attribute Name {$c},
                           for $p in //Party[.//@committee = $c]
                           return element Party {attribute Name {$p/@name},
                                                 attribute NumberOfMembers
                                                 {count($p//Assignment[@committee=$c])}
                                 }
       }
}
```

With a let clause and fewer computed constructors:

```
element Result {
 for $c in distinct-values(//@committee)
 let $parties := for $p in //Party[.//@committee = $c]
                 return <Party Name="{$p/@name}"
                               NumberOfMembers="{count($p//Assignment[@committee=$c])}"/>
 return <Committee Name="{$c}">{$parties}</Committee>
}
```

With group by:

```
element Result {
 for $a in //Assignment
 group by $c := $a/@committee
 return element Committee {attribute Name {$c},
                           for $p in $a/../..
                           return element Party {attribute Name {$p/@name},
                                                 attribute NumberOfMembers {count($a[../.. is $p])}}
         }
}
```

Clearer with ancestor and variable:

```
element Result {
  for $a in //Assignment
  group by $c := $a/@committee
  let $parties := $a/ancestor::Party
  return element Committee {attribute Name {$c},
                            for $p in $parties
                            return element Party {attribute Name {$p/@name},
                                                  attribute NumberOfMembers
                                                  {count($a[ancestor::Party is $p])}}
          }
}
```

c) What is the result of the following XQuery expression with the given sample data?
element U {distinct-values(//*[@*/name() = "name"]/child::*/child::*/@*/name())}

<U>role committee</U>

Question 5 (3 points)

Consider the XML document on the next page that only contains sample data to illustrate the structure!
Write an XSLT that presents the data as html in the following fashion:

Parties

Party	Number of MPs	Number of committee chairs	Number of committees with representation
Social democrats	4	2	5
Republicans	4	3	7
Libertarians	4	0	4
Green Party	3	2	4

With XSLT 1:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:transform version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:output method="html"/>
    <xsl:variable name="d" select="document('q45.xml')" />
    <xsl:template match="/">
        <html>
            <head><title>Question 5 with XSLT 1</title></head>
            <body>
                <h2>Parties</h2>
                <table cellpadding="5" border="1">
                    <tr>
                        <th>Party</th>
                        <th>Number of MPs</th>
                        <th>Number of committee chairs</th>
                        <th>Number of committees with representation</th>
                    </tr>
                    <xsl:apply-templates select="$d//Party"/>
                </table>
            </body>
        </html>
    </xsl:template>
    <xsl:template match="Party">
        <tr>
            <td><xsl:value-of select="@name"/></td>
            <td><xsl:value-of select="count(MP)"/></td>
            <td><xsl:value-of select="count(MP/Assignment[@role = 'chair'])"/></td>
            <td><xsl:value-of select="count(MP/Assignment/@committee[not(. =
preceding::Assignment[../../@name = current()/@name]/@committee)])"/></td>
        </tr>
    </xsl:template>
</xsl:transform>
```

A little simpler with XSLT 2:

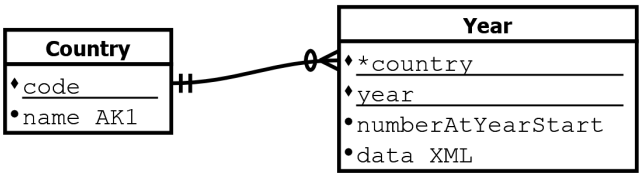
```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:transform version="2.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
   <xsl:output method="html"/>
    <xsl:template match="/">
        <html>
            <head><title>Question 5 with XSLT 2</title></head>
            <body>
                <h2>Parties</h2>
                <table cellpadding="5" border="1">
                    <tr>
                        <th>Party</th>
                        <th>Number of MPs</th>
                        <th>Number of committee chairs</th>
                        <th>Number of committees with representation</th>
                    </tr>
                    <xsl:apply-templates select=".//Party"/>
                </table>
            </body>
        </html>
    </xsl:template>
    <xsl:template match="Party">
        <tr>
            <td><xsl:value-of select="@name"/></td>
            <td><xsl:value-of select="count(MP)"/></td>
            <td><xsl:value-of select="count(MP/Assignment[@role = 'chair'])"/></td>
            <td><xsl:value-of select="count(distinct-values(MP/Assignment/@committee))"/></td>
        </tr>
    </xsl:template>
</xsl:transform>
```

XML document for question 4 and question 5

```
<?xml version="1.0" encoding="UTF-8" ?>
<ParliamentDB>
    <Party name="Social democrats">
        <MP name="Johan Löfstrand" birthyear="1976">
            <Assignment role="chair" committee="Environment committee"/>
            <Assignment role="member" committee="Finance committee"/>
        </MP>
        <MP name="Hillevi Larsson" birthyear="1974">
            <Assignment role="alternate" committee="Defense committee"/>
            <Assignment role="member" committee="Culture committee"/>
        </MP>
        <MP name="Monica Green" birthyear="1959">
            <Assignment role="chair" committee="Infrastructure committee"/>
            <Assignment role="alternate" committee="Culture committee"/>
        </MP>
        <MP name="Arhe Hamednaca" birthyear="1953"/>
    </Party>
    <Party name="Republicans">
        <MP name="Ewa Thalén Finné" birthyear="1959">
            <Assignment role="member" committee="Environment committee"/>
            <Assignment role="member" committee="Finance committee"/>
        </MP>
        <MP name="Jessika Roswall" birthyear="1972">
            <Assignment role="member" committee="Defense committee"/>
            <Assignment role="chair" committee="Justice committee"/>
        </MP>
        <MP name="Cecilia Widegren" birthyear="1973">
            <Assignment role="chair" committee="Taxation committee"/>
            <Assignment role="member" committee="Infrastructure committee"/>
            <Assignment role="member" committee="Culture committee"/>
        </MP>
        <MP name="Maria Stockhaus" birthyear="1963">
            <Assignment role="alternate" committee="Justice committee"/>
            <Assignment role="chair" committee="Defense committee"/>
        </MP>
    </Party>
    <Party name="Libertarians">
        <MP name="Roger Hedlund" birthyear="1979">
            <Assignment role="member" committee="Environment committee"/>
            <Assignment role="alternate" committee="Education committee"/>
        </MP>
        <MP name="Angelika Bengtsson" birthyear="1990">
            <Assignment role="alternate" committee="Defense committee"/>
            <Assignment role="alternate" committee="Culture committee"/>
        </MP>
        <MP name="Linus Bylund" birthyear="1978"/>
        <MP name="Dennis Dioukarev" birthyear="1993" />
    </Party>
    <Party name="Green Party">
        <MP name="Jonas Eriksson" birthyear="1967"/>
        <MP name="Emma Hult" birthyear="1988">
            <Assignment role="member" committee="Defense committee"/>
            <Assignment role="member" committee="Justice committee"/>
            <Assignment role="chair" committee="Culture committee"/>
        </MP>
        <MP name="Annika Hirvonen" birthyear="1989">
            <Assignment role="chair" committee="Finance committee"/>
            <Assignment role="member" committee="Culture committee"/>
        </MP>
    </Party>
</ParliamentDB>
```

Question 6 (2 + 2 + 2 + 2 = 8 points)

Consider the following relational database model!



Comments:
The column data is of the data type XML and accepts data that conform to the rules in the following DTD with root element PopulationChanges. All columns are defined as NOT NULL.

```
<!ELEMENT PopulationChanges (NrOfImmigrants, NrOfEmigrants, NrOfBirths, NrOfDeaths+)>
<!ELEMENT NrOfImmigrants (#PCDATA)>
<!ELEMENT NrOfEmigrants (#PCDATA)>
<!ELEMENT NrOfBirths (#PCDATA)>
<!ELEMENT NrOfDeaths (#PCDATA)>
<!ATTLIST NrOfDeaths Cause CDATA #REQUIRED>
```

There are never two NrOfDeaths with the same cause. There is always one NrOfDeaths with cause "natural". All other values (the text nodes) are integers.

Write **standard SQL** for the following:

a) Retrieve information about the total immigration to Sweden, Denmark, Finland, Norway and Iceland! The result shall have one row per year and the following columns: Year, NumberOfImmigrants.

```
SELECT year, SUM(XMLCAST(XMLQUERY('$DATA//NrOfImmigrants') AS INTEGER)) AS
NumberOfImmigrants
FROM Country, Year
WHERE code = country
AND name IN ('Sweden', 'Denmark', 'Finland', 'Norway', 'Iceland')
GROUP by year
```

Or with XMLTABLE:

```
SELECT year, SUM(nroi) AS NumberOfImmigrants
FROM Country, Year, XMLTABLE('$DATA//NrOfImmigrants'
                                    COLUMNS nroi INTEGER PATH '.')
WHERE code = country
AND name IN ('Sweden', 'Denmark', 'Finland', 'Norway', 'Iceland')
GROUP by year
```

b) Retrieve the countries where no murders (cause of death: murder) occurred in 2017! The result shall have the following structure:

```
<Result>
    <Country>
        <Code>the country code</Code><Name>the name of the country</Name>
    </Country>
    <Country>
        <Code>the country code</Code><Name>the name of the country</Name>
    </Country>
</Result>
```

```
SELECT XMLELEMENT(NAME "Result",
        XMLAGG(XMLFOREST(XMLFOREST(code AS "Code", name AS "Name") AS
"Country")))
FROM Country, Year
WHERE code = country AND year = 2017
AND NOT XMLEXISTS('$DATA//NrOfDeaths[@Cause = "murder" and text() > 0]')
```

It is possible to use XMLFOREST instead of XMLELEMENT and vice versa. The condition in the predicate can exclude the second part if we assume that the value is greater than zero if an NrOfDeaths element exists.

c) Retrieve information about countries with less immigrants than emigrants in 2019! The result shall have the following structure:

```
<Result>
    <Country code="" name="" nrOfImmigrantsIn2019=""/>
    <Country code="" name="" nrOfImmigrantsIn2019=""/>
    <Country code="" name="" nrOfImmigrantsIn2019=""/>
</Result>
```

```
SELECT XMLELEMENT(NAME "Result",
                XMLAGG(XMLELEMENT(NAME "Country",
                        XMLATTRIBUTES(code AS "code", name AS "name",
                                NumberOfImmigrants AS "nrOfImmigrantsIn2019"))))
FROM Country, Year,
    XMLTABLE('$DATA//PopulationChanges[number(NrOfImmigrants) < number(NrOfEmigrants)]'
                COLUMNS NumberOfImmigrants INTEGER PATH 'NrOfImmigrants')
WHERE code = country AND year = 2019
```

Or we can create the attribute in XQuery:

```
SELECT XMLELEMENT(NAME "Result",
                XMLAGG(XMLELEMENT(NAME "Country",
                        XMLATTRIBUTES(code AS "code", name AS "name"), xa)))
FROM Country, Year, XMLTABLE('
    for $e in $DATA//PopulationChanges[number(NrOfImmigrants) < number(NrOfEmigrants)]
    return attribute nrOfImmigrantsIn2019 {$e/NrOfImmigrants}
                        ') t(xa)
WHERE code = country AND year = 2019
```

Write **SQL Server SQL** for the following:

d) Retrieve the countries where no murders (cause of death: murder) occurred in 2017! The result shall have the following structure:

```
<Result>
    <Country code="">the name of the country</Country>
    <Country code="">the name of the country</Country>
    <Country code="">the name of the country</Country>
</Result>
```

```
SELECT code "@code", name "*"
FROM Country, Year
WHERE Country.code = Year.country AND Year.year = 2017
  AND data.exist('//NrOfDeaths[@Cause = "murder" and text() > 0]') = 0
FOR XML PATH ('Country'), ROOT ('Result')
```

The condition in the predicate can be simplified to [@Cause = "murder"] if we assume that a NrOfDeaths element would only exists if the value is higher that zero.