Exam Solution

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

2024-05-30

- 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 **question 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, XMLTEXT, 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. Well-formed XML
2. PSVI
3. processing-instruction
4. JSON Schema
5. shredding
6. SVG

1. An XML document that has exactly one root element and is syntactically correct (all subelements are nested correctly, all elements and attributes are opened and closed correctly). A well-formed XML document should begin with an XML declaration.
2. PSVI stands for Post Schema Validation Infoset and is a model for an abstract node representation of XML documents that extends the Infoset with validation information (for XML documents that are associated to a schema).
3. A special type of XML node. A processing-instruction has the following format: <?name content?>. The XML declaration looks like a processing-instruction, but it is not considered to be one. A processing-instruction (as the name implies) is used to provide information to the XML processor (how to handle or render XML).
4. Defines rules for the structure of JSON objects/structures. A JSON Schema is a JSON object itself (JSON Schema uses JSON syntax). JSON Schema can define things like properties for objects, data types and other restrictions.
5. Shredding is the process of breaking down an XML document (or other semi-structured data) into its atomic values (from text nodes and attribute nodes) most commonly in order to place them in a relational database.
6. Stands for Scalable Vector Graphics. It is an XML-based language for representing vector graphics. SVG defines elements and attributes for representing circles, rectangles, ellipses, lines, text and more and things like coordinates, dimensions, colors, transparency and fonts.

Question 2 (2 points)

Create a JSON object that conforms to the following JSON Schema!

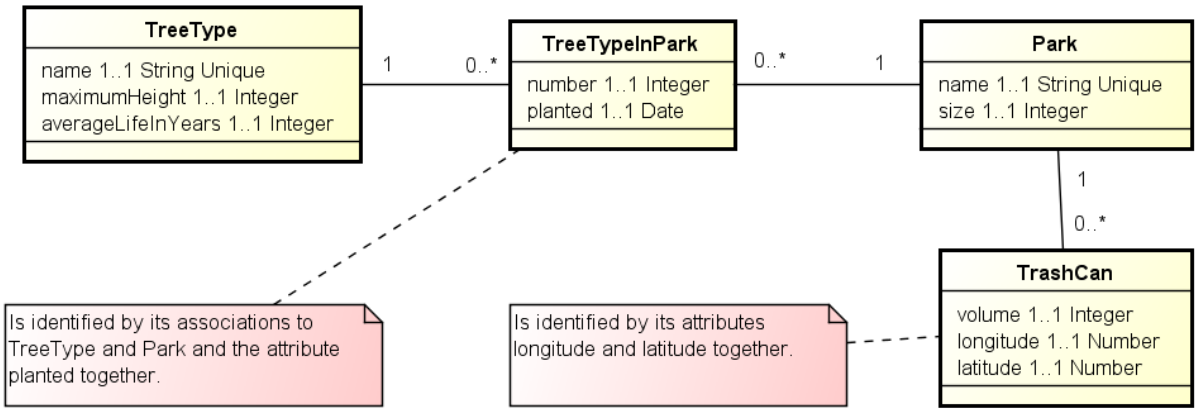
```
{
  "$schema": "https://json-schema.org/draft/2020-12/schema",
  "$id": "http://dsv.su.se/SDXML/jsonschema/product",
  "type": "object",
  "title": "A product",
  "description": "An object representation of a product",
  "properties": {
    "name": {"type": "string"},
    "colors": {
      "type": "array",
      "items": {"type": "string"},
      "minItems": 3,
```

```
        "uniqueItems": true
      },
      "sizes": {
        "type": "array",
        "items": {
          "type": "object",
          "properties": {
            "code": {"type": "string"},
            "price": {"type": "integer"}
          },
          "additionalProperties": false,
          "required": ["code", "price"]
        },
        "minItems": 2,
        "uniqueItems": true
      },
      "available": {"type": "boolean"}
    },
    "additionalProperties": false,
    "required": ["name", "colors", "sizes", "available"]
}
```

```
{
    "name": "CNN T-Shirt",
    "available": true,
    "colors": ["blue", "red", "white"],
    "sizes": [
        {
            "code": "S",
            "price": 100
        },
        {
            "code": "L",
            "price": 120
        }
    ]
}
```

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.



Comments about the attributes:
TreeType.maximumHeight is specified in meters.
TreeTypeInPark.number is always 1 or greater, and it specifies how many trees of the associated type that were planted in the associated park at the specific date.
Park.size is specified in square meters.
TrashCan.volume is specified in liters.

XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<Data>
    <Park name="Green Hill Park" size="1200"/>
    <Park name="Sea Side Park" size="1200">
        <TrashCan volume="100" longitude="17.940987" latitude="59.407562"/>
    </Park>
    <Park name="Bear Tree Park" size="400">
        <Planting treeType="1" number="15" date="2020-04-05"/>
        <Planting treeType="2" number="4" date="2020-04-05"/>
        <Planting treeType="2" number="2" date="2021-04-06"/>
    </Park>
    <Park name="Coral Bay Park" size="970">
        <TrashCan volume="100" longitude="17.931878" latitude="59.407562"/>
        <TrashCan volume="100" longitude="17.931878" latitude="59.407504"/>
        <Planting treeType="2" number="12" date="2020-08-10"/>
    </Park>
    <!-- More Park elements with zero or more TrashCan elements followed by zero or more Planting elements -->
    <TreeType id="1" name="Palm" maximumHeight="21" averageLifeInYears="50"/>
    <TreeType id="2" name="Maple" maximumHeight="37" averageLifeInYears="130"/>
    <TreeType id="3" name="Pine" maximumHeight="80" averageLifeInYears="120"/>
    <TreeType id="4" name="Cherry" maximumHeight="24" averageLifeInYears="100"/>
    <TreeType id="5" name="Oak" maximumHeight="30" averageLifeInYears="200"/>
    <!-- More TreeType elements -->
</Data>
```

XML Schema:

```
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema">
    <element name="Data">
        <complexType>
            <sequence>
                <element name="Park" type="ParkType" minOccurs="0" maxOccurs="unbounded"/>
                <element name="TreeType" minOccurs="0" maxOccurs="unbounded">
                    <complexType>
                        <attribute name="id" type="integer" use="required"/>
                        <attribute name="name" type="string" use="required"/>
                        <attribute name="maximumHeight" type="integer" use="required"/>
                        <attribute name="averageLifeInYears" type="integer" use="required"/>
                    </complexType>
                </element>
            </sequence>
        </complexType>
    </element>
    <complexType name="ParkType">
        <sequence>
            <element name="TrashCan" minOccurs="0" maxOccurs="unbounded">
                <complexType>
                    <attribute name="volume" type="integer" use="required"/>
                    <attribute name="longitude" type="integer" use="required"/>
                    <attribute name="latitude" type="integer" use="required"/>
                </complexType>
            </element>
            <element name="Planting" minOccurs="0" maxOccurs="unbounded">
                <complexType>
                    <attribute name="treeType" type="integer" use="required"/>
                    <attribute name="number" type="integer" use="required"/>
                    <attribute name="date" type="date" use="required"/>
                </complexType>
            </element>
        </sequence>
        <attribute name="name" type="string" use="required"/>
        <attribute name="size" type="integer" use="required"/>
    </complexType>
</schema>
```

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

Consider the XML document on the last page (after question 6) that only contains sample data in order to illustrate the structure.

a) Write XQuery for the following:
Retrieve information about every movie per genre according to the following structure!

```
<Genres>
    <Genre Name="">
        <Movie Title="" NumberOfOtherGenres=""/>
        <Movie Title="" NumberOfOtherGenres=""/>
        <Movie Title="" NumberOfOtherGenres=""/>
    </Genre>
    <Genre Name="">
        <Movie Title="" NumberOfOtherGenres=""/>
        <Movie Title="" NumberOfOtherGenres=""/>
    </Genre>
</Genres>
```

```
element Genres {
 for $g in distinct-values(//Genre)
 let $movies := for $m in //Movie[Genre = $g]
                return element Movie {attribute Title {$m/@title},
                                      attribute NumberOfOtherGenres {count($m/Genre) - 1}}
 return element Genre {attribute Name {$g}, $movies}
}
```

```
element Genres {
 for $genre in //Genre
 group by $g := $genre/text()
 let $movies := for $m in $genre/..
                return element Movie {attribute Title {$m/@title},
                                      attribute NumberOfOtherGenres {count($m/Genre) - 1}}
 return element Genre {attribute Name {$g}, $movies}
}
```

b) Write XQuery for the following:
Retrieve information about all the showings according to the following structure!

```
<Result>
    <Genre name="">
        <Showing starttime="" movie="" cinema="" hall=""/>
        <Showing starttime="" movie="" cinema="" hall=""/>
        <Showing starttime="" movie="" cinema="" hall=""/>
    </Genre>
    <Genre name="">
        <Showing starttime="" movie="" cinema="" hall=""/>
        <Showing starttime="" movie="" cinema="" hall=""/>
    </Genre>
</Result>
```

```
element Result {
 for $g in distinct-values(//Genre)
 let $movies := for $s in //Movie[Genre = $g]//Showing
                  return element Showing {$s/@starttime, $s/@hall,
                                          attribute movie {$s/../../@title},
                                          attribute cinema {$s/../@name}}
 return element Genre {attribute Name {$g}, $movies}
}

element Result {
 for $genre in //Genre
 group by $g := $genre/text()
 let $movies := for $s in $genre/..//Showing
                  return element Showing {$s/@starttime, $s/@hall,
                                          attribute movie {$s/../../@title},
                                          attribute cinema {$s/../@name}}
 return element Genre {attribute Name {$g}, $movies}
}

<Result>{
 for $g in distinct-values(//Genre)
 let $movies := for $s in //Movie[Genre = $g]//Showing
                return <Showing movie="{$s/../../@title}" cinema="{$s/../@name}">
                          {$s/@starttime, $s/@hall}
                       </Showing>
 return <Genre Name="{$g}">{$movies}</Genre>
}</Result>
```

c) What is the result of the following XQuery expression with the given sample data?
element D {(//text()/ancestor::*[not(@*)]/name())[position() < 4]}

```
<D>MoviesToday Genre Genre</D>
```

Question 5 (3 points)

Consider the XML document on the last page (after question 6) that only contains sample data to illustrate the structure!
Write an XSLT that presents the data as html in the following fashion:

Cinemas

Cinema	Number of halls	Number of showings	Number of different movies shown
Saga	3	7	3
Scandinavia	2	5	3
Grand	3	8	4

Solution with XSLT 1:

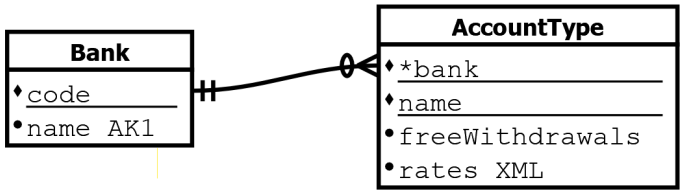
```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:transform version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:output method="html"/>
    <xsl:variable name="d" select="document('q45.xml')" />
    <xsl:template match="/">
        <html>
            <body>
                <h2>Cinemas</h2>
                <table cellpadding="5" border="1">
                    <tr>
                        <th>Cinema</th>
                        <th>Number of halls</th>
                        <th>Number of showings</th>
                        <th>Number of different movies shown</th>
                    </tr>
                    <xsl:apply-templates select="$d//Cinema
                                                    [not(@name = preceding::Cinema/@name)]/@name"/>
                </table>
            </body>
        </html>
    </xsl:template>
    <xsl:template match="@name">
        <xsl:variable name="c" select="$d//Cinema[@name = current()]"/>
        <tr>
            <td><xsl:value-of select="."/></td>
            <td><xsl:value-of select="count($c/Showing[not(@hall =
                                            preceding::Showing[../@name = current()]/@hall)])"/></td>
            <td><xsl:value-of select="count($c/Showing)"/></td>
            <td><xsl:value-of select="count($c)"/></td>
        </tr>
    </xsl:template>
</xsl:transform>
```

Solution with XSLT 3:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:transform version="3.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform" expand-text="yes">
    <xsl:output method="html"/>
    <xsl:template match="/">
        <html>
            <body>
                <h2>Cinemas</h2>
                <table cellpadding="5" border="1">
                    <tr>
                        <th>Cinema</th>
                        <th>Number of halls</th>
                        <th>Number of showings</th>
                        <th>Number of different movies shown</th>
                    </tr>
                    <xsl:for-each-group select="//Showing" group-by="../@name">
                        <tr>
                            <td>{current-grouping-key()}</td>
                            <td>{count(distinct-values(current-group()/@hall))}</td>
                            <td>{count(current-group())}</td>
                            <td>{count(current-group()/ancestor::Movie)}</td>
                        </tr>
                    </xsl:for-each-group>
                </table>
            </body>
        </html>
    </xsl:template>
</xsl:transform>
```

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

Consider the following relational database model!



Comments:
The column AccountType.rates is of the data type XML and accepts data that conform to the rules in the following DTD with root element Data. The column AccountType.freeWithdrawals is BOOLEAN. All other columns are STRING. All columns are defined as NOT NULL.

```
<!ELEMENT Data (Period+)>
<!ELEMENT Period (InterestRate+)>
<!ATTLIST Period StartDate CDATA #REQUIRED EndDate CDATA #IMPLIED >
<!ELEMENT InterestRate EMPTY>
<!ATTLIST InterestRate MinimumBalance CDATA #REQUIRED Rate CDATA #REQUIRED>
```

There is always at most one Period without an EndDate and no two periods may overlap. No two InterestRate elements inside the same Period may have the same MinimumBalance. The values of the attributes StartDate and EndDate are always valid dates. The values of MinimumBalance and Rate are always valid numbers.

Write **standard SQL** for the following:

a) Retrieve information about the highest ever interest rate per bank! The result shall have one row per bank and the following columns: Code, Name, HighestRateEver.

```
SELECT Code, Name, (SELECT XMLQUERY('max($a//@Rate)' PASSING XMLAGG(rates) AS "a")
                    FROM AccountType WHERE bank = code) AS HighestRateEver
FROM Bank
```

Or with XMLTABLE (OUTER JOINS just in case there are banks without account types):

```
SELECT Code, Bank.Name, MAX(rate) AS HighestRateEver
FROM Bank
     LEFT OUTER JOIN AccountType ON bank = code
     LEFT OUTER JOIN XMLTABLE('$a//@Rate' PASSING rates AS "a"
                              COLUMNS rate REAL PATH '.') ON 1=1
GROUP BY Code, Bank.Name
```

Or one max per account type and one max per bank:

```
SELECT Code, Bank.Name,
     MAX(XMLCAST(XMLQUERY('max($a//@Rate)' PASSING rates AS "a") AS REAL))
     AS HighestRateEver
FROM Bank LEFT OUTER JOIN AccountType ON bank = code
GROUP BY Code, Bank.Name
```

It is also possible to use the function COALESCE if we want to get 0 instead of NULL in the result.

Solutions that exclude banks that have no account types are also accepted.

b) Retrieve information about all the account types that offer free withdrawals! The result shall have the following structure:

```
<Result>
    <Bank Name="" Code="">
        <AccountType Name=""/>
    </Bank>
    <Bank Name="" Code="">
        <AccountType Name=""/>
        <AccountType Name=""/>
    </Bank>
</Result>
```

```
SELECT XMLFOREST(XMLAGG(XMLELEMENT(NAME "Bank",
                                   XMLATTRIBUTES(code AS "Code",
                                                 name AS "Name"), typesXML)) AS "Result")
FROM (SELECT Code, Bank.Name,
             XMLAGG(XMLELEMENT(NAME "AccountType",
                               XMLATTRIBUTES(AccountType.Name AS "Name"))) AS typesXML
      FROM Bank INNER JOIN AccountType ON bank = code
      WHERE freeWithdrawals
      GROUP BY Code, Bank.Name)
```

c) Which account types offer currently at least 4% in interest rate if you have a balance of 50000? The result shall have the following structure:

```
<Result>
    <AccountType bankcode="" bankname="" name=""/>
    <AccountType bankcode="" bankname="" name=""/>
    <AccouctType bankcode="" bankname="" name=""/>
</Result>
```

This solution assumes that the interest rate only increases when the balance increases

```
SELECT XMLFOREST(XMLAGG(XMLELEMENT(NAME "AccountType",
                                   XMLATTRIBUTES(Bank.name AS "bankname",
                                                 Bank.code AS "bankcode",
                                                 AccountType.Name AS "name"))
                        ) AS "Result")
FROM Bank INNER JOIN AccountType ON bank = code
WHERE XMLEXISTS('$RATES//Period[not(@EndDate)]/InterestRate
                                   [@MinimumBalance <= 50000 and @Rate >= 4]' PASSING rates)
```

The following solution assumes that the interest rate can decrease when the balance increases

```
SELECT XMLFOREST(XMLAGG(XMLELEMENT(NAME "AccountType",
                                   XMLATTRIBUTES(Bank.name AS "bankname",
                                                 Bank.code AS "bankcode",
                                                 AccountType.Name AS "name"))
                        ) AS "Result")
FROM Bank INNER JOIN AccountType ON bank = code
WHERE XMLEXISTS('let $irs := $RATES//Period[not(@EndDate)]/InterestRate
                                                 [@MinimumBalance <= 50000]
                 return $irs[@MinimumBalance = max($irs/@MinimumBalance) and @Rate >= 4]'
                 PASSING rates)
```

Here, the condition is expressed in a different way in a predicate

```
SELECT XMLFOREST(XMLAGG(XMLELEMENT(NAME "AccountType",
                                   XMLATTRIBUTES(Bank.name AS "bankname",
                                                 Bank.code AS "bankcode",
                                                 AccountType.Name AS "name"))
                        ) AS "Result")
FROM Bank INNER JOIN AccountType ON bank = code
WHERE XMLEXISTS('$RATES//Period[not(@EndDate)]/InterestRate
                        [@MinimumBalance <= 50000 and @Rate >= 4 and
                        not(@MinimumBalance < ../InterestRate/@MinimumBalance[. <= 50000])]'
                        PASSING rates)
```

It may be necessary to use xs:integer(...) or .../number() to avoid comparing the values of the attribute MinimumBalance as strings and then getting 2000 being greater than 10000 (because "2000" is greater than "10000"). This is not a requirement for submitted solutions.

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

d) Retrieve information about each bank according to the following structure:

```
<Result>
    <Bank code="" name="" numberOfActiveAccountTypes=""/>
    <Bank code="" name="" numberOfActiveAccountTypes=""/>
</Result>
```

Definition: An account type is considered to be active if it has a rate period without an end date.

```
SELECT code, name, (SELECT COUNT(*)
                    FROM AccountType
                    WHERE bank = code
                      AND rates.exist('//Period[not(@EndDate)]') = 1)
                                                    AS numberOfActiveAccountTypes
FROM Bank
FOR XML RAW('Bank'), ROOT('Result')
```

Or with a JOIN (with a weird condition to avoid contradicting the OUTER JOIN)

```
SELECT code, Bank.name, COUNT(bank) AS numberOfActiveAccountTypes
FROM Bank LEFT OUTER JOIN AccountType
                          ON bank = code AND rates.exist('//Period[not(@EndDate)]') = 1
GROUP BY code, Bank.name
FOR XML RAW('Bank'), ROOT('Result')
```

Or with a nested statement to avoid the, perhaps inappropriate, join condition of the previous solution

```
SELECT code, name, COUNT(bank) AS numberOfActiveAccountTypes
FROM Bank LEFT OUTER JOIN
        (SELECT bank
        FROM AccountType
        WHERE rates.exist('//Period[not(@EndDate)]') = 1) AT ON bank = code
GROUP BY code, name
FOR XML RAW('Bank'), ROOT('Result')
```

All solutions that only include banks with at least one active account type are also accepted.

XML document for question 4 and question 5

```
<?xml version="1.0" encoding="UTF-8" ?>
<MoviesToday>
    <Movie title="Civil War" length="109">
        <Genre>Action</Genre>
        <Genre>Thriller</Genre>
        <Cinema name="Saga">
            <Showing starttime="14:30" hall="1" ticketprice="110"/>
            <Showing starttime="16:30" hall="2" ticketprice="90"/>
            <Showing starttime="17:50" hall="1" ticketprice="110"/>
        </Cinema>
        <Cinema name="Scandinavia">
            <Showing starttime="16:00" hall="1" ticketprice="110"/>
            <Showing starttime="18:20" hall="1" ticketprice="120"/>
        </Cinema>
        <Cinema name="Grand">
            <Showing starttime="22:10" hall="1" ticketprice="120"/>
        </Cinema>
    </Movie>
    <Movie title="Furiosa" length="158">
        <Genre>Action</Genre>
        <Cinema name="Saga">
            <Showing starttime="14:40" hall="3" ticketprice="110"/>
            <Showing starttime="18:00" hall="3" ticketprice="120"/>
            <Showing starttime="21:50" hall="1" ticketprice="120"/>
        </Cinema>
        <Cinema name="Scandinavia">
            <Showing starttime="17:00" hall="2" ticketprice="120"/>
            <Showing starttime="19:50" hall="2" ticketprice="120"/>
        </Cinema>
        <Cinema name="Grand">
            <Showing starttime="16:00" hall="1" ticketprice="120"/>
            <Showing starttime="19:20" hall="1" ticketprice="120"/>
        </Cinema>
    </Movie>
    <Movie title="Abigail" length="94">
        <Genre>Comedy</Genre>
        <Genre>Horror</Genre>
        <Genre>Fantasy</Genre>
        <Cinema name="Saga">
            <Showing starttime="14:20" hall="2" ticketprice="90"/>
        </Cinema>
        <Cinema name="Scandinavia">
            <Showing starttime="21:00" hall="1" ticketprice="100"/>
        </Cinema>
        <Cinema name="Grand">
            <Showing starttime="16:00" hall="3" ticketprice="90"/>
            <Showing starttime="18:20" hall="3" ticketprice="100"/>
            <Showing starttime="21:10" hall="3" ticketprice="100"/>
        </Cinema>
    </Movie>
    <Movie title="Whitney" length="97">
        <Genre>Drama</Genre>
        <Genre>Music</Genre>
        <Cinema name="Grand">
            <Showing starttime="15:20" hall="2" ticketprice="100"/>
            <Showing starttime="22:10" hall="2" ticketprice="100"/>
        </Cinema>
    </Movie>
</MoviesToday>
```