CompSci 316 Fall 2018: Homework #3

100 points (8.75% of course grade) + 10 points extra credit
Assigned: Thursday, October 18
Due: Tuesday, November 6

This homework should be done in parts as soon as relevant topics are covered in lectures. If you wait until the last minute, you might be overwhelmed.

For Problem 1, you will need to use Gradiance. Access Gradiance via the "Gradiance" link on the course website. There is no need to turn in anything else for this problem; your score will be tracked automatically.

For other problems, you will need to turn in the required files electronically. Please read the "Help → Submitting Non-Gradiance Work" section of the course website, and follow the submission instructions for each problem carefully.

Problems 2, 3, X1, and X2 should be completed on your course VM. Before you start, make sure you refresh your VM, by logging into your VM and issuing the following command:

```
/opt/dbcourse/sync.sh
```

Problem 1 (15 points)

Complete the Gradiance homework titled "Homework 3.1 (XML)."

Problem 2 (60 points)

In `/opt/dbcourse/examples/congress/` on your VM, you will find an XML file `congress.xml` containing information about the current US Congress. Logically, the file consists of two sections:

- Each `person` element under `congress/people` stores information about a legislator, including the roles he or she has served in the Congress. A `role` with type "`rep`" indicates a Representative (member of the House), while a `role` with type "`sen`" indicates a Senator (member of the Senate). A `role` is current if its `current` attribute equals 1.
- Each `committee` element under `congress/committees` stores information about a committee. It has a list of members, whose ids reference those of `person` elements in the first section; `role` specifies the role of the member in the committee (e.g., chair or ranking member). Oftentimes a committee can have subcommittees. Each `subcommittee` element has its own list of members, which should be a subset of the committee members. A legislator can serve on multiple committees, and even multiple subcommittees under the same committee.

Write queries in XQuery to answer the following questions. Unless otherwise noted, please make sure that your answer appears under a single root element `<result>`. For each question below, say (a), write your XQuery in a file named `a.xq`, (replace "`a`" with "`b`", "`c`", and other parts as appropriate). You can run your query as follows:

```
saxonb-xquery -s /opt/dbcourse/examples/congress/congress.xml a.xq
```

Do not hardcode the input file using the `doc("…")` function in your XQuery; instead, specify the input file using a command-line option as above. Submit all your query (`.xq`) files. (There is no need to respond to questions enclosed in parentheses below.)

Please refer to the document "XML Tips" on the course Web site for additional instructions on running `saxonb-xquery`, the Saxon XQuery processor. Because Saxon does not use any indexes and does not have a sophisticated optimizer, query performance may be heavily influenced by the way you write your queries. If a particular query takes forever to run, consider reordering loops and evaluating selections (filters) as early as possible. Note that you can add comments to your queries by enclosing them in "`(:`" and "`:)`".

(a) Find the legislator(s) with last name "`Price`". Simply print the entire `person` element(s). Use `ends-with(str1, str2)` to test if `str1` ends with `str2`. (Do you know that he was a Duke professor?)

(b) Find who serves the role of "`Chairman`" on the Senate Committee on the Judiciary, code name "`SSJU`". Simply print the entire `person` element. (Hasn't he been in the news recently?)

(c) List all current female Senators born since 1960. Format each of them as an element of the form `<senator name="`*name*`"/>`. Order them by the name attribute. You can use `xs:date("1939-12-31") >= xs:date("2000-01-01")` to test if the date 1939-12-31 is the same as or later than the date 2000-01-01.

(d) List, for each current Senator of NC, their name, current party, and date when they first became a Senator. Format each Senator as an element of the form `<senator first_start="`*date*`" party="`*party*`">`*name*`</senator>`. Order them by seniority in the Senate; i.e., the one with the earlier start date should be listed first. (Is anyone up for reelection?)

(e) List the names of current Representatives who at some point earlier also served as Senators. Format each of them as an element of the form `<member>`*name*`</member>`. Order them by name. (This is rather uncommon, but once upon a time there was a President who had done this!)

(f) List the names of legislators who are NOT serving in any committee or subcommittee. Format each of them as an element for the form `<person>…</person>`. (By the way, do you know why they aren't?)

(g) Find the number of current legislators for each party by gender. Your output should look like the following (whitespace is unimportant):

```
<result>
  <Democrat><M count="…"/><F count="…"/></Democrat>
  <Republican><M count="…"/><F count="…"/></Republican>
  <Independent><M count="…"/><F count="…"/></Independent>
</result>
```

To specify computed values (expressions) as output element/attribute names, you can use the following alternative XQuery syntax for constructing elements/attributes:

```
… return element {$computed_etag} {       (: with {}, tag name is computed :)
    attribute {concat('attr', '1')} {$computed_aval},
    attribute attr2 {$computed_aval2},    (: without {}, attr2 becomes the
    …                                        attribute name verbatim :)
  } …
```

Problem 3 (25 points)

Now consider the exact same database from Problem 2, now stored as JSON documents in a MongoDB database. To start the MongoDB database server and construct this database named congress, use the following commands in your VM:

```
sudo service mongod start
mongorestore --db congress /opt/dbcourse/examples/congress/mongodb-dump/congress
```

The database contains two types of documents inside two collections, `people` and `committees`. To see these documents, use the following commands:

```
mongo --quiet congress --eval 'db.people.find({}).toArray()'
mongo --quiet congress --eval 'db.committees.find({}).toArray()'
```

The structures of these documents are self-explanatory and resemble those of `<person>` and `<committee>` elements in Problem 2, but beware of the subtleties due to differences between XML and JSON. Note that the documents are identified by their `_id` attribute values, which are person ids and committee codes, respectively.

Write MongoDB queries (in MongoDB shell syntax) to answer the following questions. Unless otherwise noted, please make sure that your answer appears as an array. Your query should have the form `db.`*`collection`*`.`*`method`*`(…).toArray()`, where *`collection`* is one of `people` and `committees`, and *`method`* is one of `find` and `aggregate`. For each question below, say (a), write your MongoDB query in a file named `a.js`, (replace "`a`" with "`b`" and "`g`" as appropriate). You can run your query as follows:

```
mongo --quiet congress < a.js
```

Submit all your query (`.js`) files. Please refer to the document "MongoDB Tips" on the course Web site for additional instructions on running and querying MongoDB.

(a) Find the legislator(s) with last name "`Price`". Simply print the entire person documents. You can use *`attr`*`:/`*`pattern`*`/` match the value of *`attr`* against a regular expression *`pattern`*. In particular, `/ XYZ$/` (note the space before `XYZ`) ensures that the string ends with a space followed by "XYZ"; "`$`" in the pattern matches the end of the string.

(b) Find who serves the role of "`Chairman`" on the Senate Committee on the Judiciary, code name "`SSJU`". Simply print the entire person document.

(g) Find the number of current legislators for each party by gender. Your output should look like the following (whitespace and ordering are unimportant):

```
[ { "count" : …, "party" : "Republican", "gender" : "F" },
  { "count" : …, "party" : "Democrat", "gender" : "M" },
  … … ]
```

Extra Credit Problem X1 (5 points)

Continuing from Problem 3, write MongoDB queries to answer the following questions. Submit all your query (`.js`) files.

(c) List all current female Senators born since 1960. Format each of them simply as `{ "name":… }`, and order them by name. You can use *`attr`*`: { $gte: ISODate("2000-01-01") }` to test if the value of *`attr`* is the same as or later than the date 2000-01-01.

(d) List, for each current Senator of NC, their name, current party, and date when they first became a Senator. Format each Senator as `{ "name":…, "first_start":…, "party":… }`, and order them by seniority in the Senate; i.e., the one with the earlier start date should be listed first.

(e) List the names of current Representatives who at some point earlier also served as Senators. Format each of them as `{ "name":… }` and order them by name.

(f) List the names of legislators who are NOT serving in any committee or subcommittee. Format each of them as `{ "name":… }`.

Extra Credit Problem X2 (5 points)

Continuing from Problem 2, your job is to produce an output XML file `percom.xml`, which presents information about legislators and their committee assignments in a more concise and readable form. The output file should be structured as follows, and conform to the DTD in `/opt/dbcourse/examples/congress/percom.dtd`.

- The root element is `congress`.
- `congress` has two child elements: `house` and `senate`, each listing its current legislators. See the description of `congress.xml` above for how to determine who are current members of the two chambers.
- Each legislator is represented as a `person` element, with a name attribute whose value is taken from `person/@name` in `congress.xml`. Under `person`, list each committee that this legislator serves in as a `committee` element. A `committee` element has a `name` attribute whose value is taken from `committee/@displayname` in `congress.xml`; it also has a `role` attribute whose value is taken from `member/@role` (or simply "`Member`" if no role is specified). Under `committee`, list each subcommittee of the committee that this legislator serves in, as a `subcommittee` element. Like a `committee` element, a `subcommittee` has a `name` attribute and a `role` attribute.

For example, here is a snippet of the output showing the committee assignment for Lisa Murkowski:

```
<?xml version="1.0" encoding="UTF-8"?>
<congress>
  <house>
    …
  </house>
  <senate>
    …
    <person name="Lisa Murkowski">
       <committee role="Member" name="Senate Committee on Indian Affairs"/>
       <committee role="Member" name="Senate Committee on Appropriations">
          <subcommittee role="Member" name="Commerce, Justice, Science, and Related Agencies"/>
          <subcommittee role="Member" name="Energy and Water Development"/>
          <subcommittee role="Member" name="Department of Homeland Security"/>
          <subcommittee role="Chairman"
                        name="Department of the Interior, Environment, and Related Agencies"/>
          <subcommittee role="Member"
                        name="Military Construction and Veterans Affairs, and Related Agencies"/>
          <subcommittee role="Member" name="Department of Defense"/>
       </committee>
       <committee role="Chairman" name="Senate Committee on Energy and Natural Resources">
          <subcommittee role="Ex Officio" name="Water and Power"/>
          <subcommittee role="Ex Officio" name="National Parks"/>
          <subcommittee role="Ex Officio" name="Energy"/>
```

```
         <subcommittee role="Ex Officio" name="Public Lands, Forests, and Mining"/>
      </committee>
      <committee role="Member" name="Senate Committee on Health, Education, Labor, and Pensions">
         <subcommittee role="Member" name="Primary Health and Retirement Security"/>
         <subcommittee role="Member" name="Children and Families"/>
      </committee>
   </person>
   …
  </senate>
</congress>
```

To generate `percom.xml` from `congress.xml`, you have several options. You can write an XQuery, a Python program using the DOM API (`xml.dom`), or any other language or API. Your code should handle any potential dangling references (e.g., ids that refer to non-existent person elements, which may arise when legislators leave and committee information becomes outdated). Please refer to the document "XML Tips" on the course website for instructions on working with XML. You should validate your output file `percom.xml` against the provided `percom.dtd`, using the following command (more information on `xmllint` can be found in "XML Tips"):

```
xmllint --dtdvalid /opt/dbcourse/examples/congress/percom.dtd --noout percom.xml
```

Submit your source code and XML output.