University of Toronto
CSC**343, Fall 2012**

Assignment 4

Due date: Sunday, 2 December at 11:59pm

1 Introduction

The MarkUs system that is sometimes used for assignment submission has a feature that allows students to form teams, if permitted by the course instructor. Imagine a more elaborate version which allows the instructor to specify what he or she considers to be desirable groups, such as groups with students from the same part of the city, with diverse majors, and with at least one group member who knows Java. The system could survey the students for the relevant information and then put them into groups according to the instructor's criteria. To get a feel for the problem domain, look at the sample data in file `Data.txt`, available on the Assignments page of the course website. A system for grouping student could store this data in XML files. In this assignment:

- You will design DTDs for XML files that can store this sort of data.
- You will create XML files that hold the sample data in `Data.txt` and are valid with respect to your DTDs.
- You will write queries on these XML files.

2 Define DTDs [18 Marks]

Below are the specifications for three DTDs you will write. You must ensure that everything that is described below can be expressed in an XML file that conforms to your DTD. There may be things that should be *disallowed* but that you cannot disallow because of the limitations of DTDs. That's okay. Express as many of the restrictions as you can.

For each DTD, you will have many options about how to structure the data. Keep in mind that redundant data is not a good thing. When deciding whether to use an attribute or a element, consider using an element for what feels like the core information and an attribute for additional information about it. (Yes, this is not at all precise.) Don't make the tedium of data entry a big design consideration. Assume that there would be a GUI that would do all the tedious parts of creating the xml files; a human should only have to enter the real content (question text etc.).

In a file called **survey.dtd**, define a DTD that specifies the format for an xml file containing a survey. It must support the following features:

- A survey has an ID, a title, and one or more questions.
- There are several types of multiple choice question: traditional multiple choice questions (where exactly one answer must be selected), multiple choice questions where any number of answers may be selected, and multiple choice questions where exactly k answers must be selected (k must not be greater than the number of answer options).
- Any multiple choice question has answer options, and there are always at least two.
- In addition to multiple choice questions there are numeric and timetable questions,

- A numeric question requires an integer answer; we won't handle floats. A numeric question may have a lower and/or upper bound on acceptable response values.
- Timetable questions have one or more timeslots associated with them (each described by text such as "Monday evenings" or "Dec 1st, 2-3pm"). For each one, the user will indicate their availability. A timetable question can offer the user two choices (indicating that they are available or not available) or these plus a third option that is in-between.
- Every question has a question ID and question text.
- Each question is either mandatory (users are required to answer these) or optional.

In a file called **responses.dtd**, define a DTD that specifies the format for an xml file containing each student in the class and their responses to the survey. It must must support the following features:

- It contains at least one student's responses to the survey.
- A student's response records the student's ID, which is a string.
- For each question on the survey that the student answered, the student's response stores the answer the student gave. A student may not have answered the optional questions, but did answer all the mandatory ones (otherwise, their responses to the survey were not recorded).
- Some students in the class may not have completed the survey. Even though there are no responses to record for such students, their student ID is still recorded in the class file.

In a file called **division.dtd**, define a DTD that specifies the format for an xml file containing a "division" of the class into groups. It must support the following features:

- A division has an ID. Optionally, it may record the minimum and maximum group size permitted.
- A division contains one or more groups, each of which has a group ID and a set of students.
- Each group has a leader, which is one of the students in the group.
- Every student in a division is in the class.

Don't get distracted by wondering whether students can answer a question multiple times, what would happen in the user-interface if they didn't answer a mandatory question, or other aspects of taking the survey. Your only responsibility is to record the relevant data.

3 Create Valid XML Instance Documents [18 Marks]

Create a valid instance document for each of your three DTDs, using the data in `Data.txt`, which is available on the Assignments page of the course website. Each of these instance documents will be separate from its DTD; in other words, the DTD will not be embedded in the XML file for the instance document.

Order matters in an XML document. Put the components of the data in the same order in your XML instance documents as they are in the data file we are providing.

Name your files survey.xml, responses.xml and division.xml. Run xmllint on your files to confirm that they are valid with respect to your DTDs.

4 Query XML Documents [64 Marks]

Write queries in XQuery to produce the following results:

1. For every student in the class, the student ID and their college. If the student has not answered the college question, give "no college reported" as their college.
2. The average group size.
3. The college that most students are in. If there is a tie, report all colleges that tied.
4. For each option (to a multiple choice question) that no one picked, the question ID and the option text.
5. The group node of every group whose leader is junior to (in an earlier year in school) some other member of that group.
6. Valid xml (satisfying the DTD in file years.dtd, available on the Assignments page) that contains the student ID and year of every student in the class.
7. Valid xml (satisfying the DTD in file histogram.dtd, available on the Assignments page) that contains a histogram showing the number of students who picked each project domain in survey.xml.
8. Valid xml (satisfying the DTD in file availability.dtd, available on the Assignments page) that shows for each group, the number of timeslots to which all group members answered yes.

Store each query in a separate file, and call these q1.xq through q8.xq.

For the queries that generate xml output, generate only the xml elements. Don't try to prepend that with the "header" information that belongs at the top of the file.

We will not be autotesting your code, so some of these questions above are not as specific about format as they would have to be if we were autotesting. The exceptions are queries where you are to produce an XML file that conforms to a DTD. These DTDs will be posted on the Assignments page. In all cases, white space is up to you. Just try to make your output readable.

You may hard-code your queries only where necessary. For example, for query 7 you will need to refer to the survey question about project domains. You can hard-code the question ID into you query, but do not hard-code in what you know to be the answer options. Your query should work if we change the number of answer options or the content of them.

Here are a few XQuery tips that may help

- Although we spent most of our time on FLWOR expressions, there are other kinds of expression. We've studied `if` expressions, `some` expressions and `every` expressions. And remember that a path expression is an expression in XQuery also.
- XQuery is an expression language. Each query is an expression, and we can nest expressions arbitrarily.
- You can put more than one expression in the `return` of a FLWOR expression if you put commas between them and enclose them in round brackets.

- XQuery is very "fiddley". It's easy to write a query that is very short, yet full of errors. And the errors can be difficult to find. There is no debugger, and the syntax errors you'll get are not as helpful as you might wish. A good way to tackle these queries is to start incredibly small and build up your final answer in increments, testing each version along the way. For example, if you need to do the equivalent of a "join" between the survey and the responses, you could start by iterating through just one of them; then make a nested loop that makes all pairs; then add on a condition that keeps only the sensible pairs. Save each version as you go. You will undoubtedly extend a query a little, break it, and then ask yourself "how was it before I broke it?"

5 Capture your results

Capture your results by running the script `runall.sh` found on the course website. It runs xmllint on each of your xml files, and galax-run on each of your queries and sends the output to a file called results.txt. If you are not able to complete a query, you may comment it out of the script.

6 Electronic Submission Instructions

You should submit your the following files electronically using the submit command in CDF:

- Information about your team (whether it is a team of one or two students): team.txt
- Your three DTD files: survey.dtd, responses.dtd, and division.dtd
- Your three XML file: survey.xml, responses.xml, and division.xml
- Your query files: q1.xq through q8.xq
- Your "capture" script: runall.sh (even if you didn't change it)
- Your results file: results.txt

You may work at home, but you must make sure that your code runs on the CDF machines.

When you have completed the assignment, move or copy your files in a directory (e.g., assignment4), and use the following command to electronically submit your files within that directory:

% submit -c csc343h -a A4 team.txt survey.dtd responses.dtd division.dtd survey.xml responses.xml division.xml q1.xq q2.xq q3.xq q4.xq q5.xq q6.xq q7.xq q8.xq runall.sh results.txt

You can also submit the files individually after you complete each part of the assignment - simply execute the submit command and give the filename that you wish to submit. You may submit your solutions as many times as you wish prior to the submission deadline (you might need to use the -f flag of the submit command). Make sure you name your files exactly as stated (including lower/upper case letters). Failure to do so will result in a mark of 0 being assigned.

Once you have submitted, be sure to check that you have submitted the correct version of each file; new or missing files will not be accepted after the due date. You may check the status of your submission using the command

% submit -l -N A4 csc343h

where -l is a hyphen followed by the letter 'ell'.