University of Toronto
CSC**343S, Winter 2011**

Assignment 3

Due: Thursday 07 April, at 10:00 pm sharp!

IMPORTANT: Read the discussion board for any updates regarding this assignment.

The parts of this assignment can be done in any order. Grace points, if you still have them, can indeed be used on this assignment. Be sure to review the course policy for grace days and late assignments.

Part 1: XML and XQuery

The Gradiance online system that you've been using this term has a large bank of questions on various topics. A course instructor picks from these to create a homework quiz. The answers of students in the class are recorded by Gradiance so that the instructor can download them and assign grades.

For this part of the assignment, you will design XML files for those components of a system like Gradiance (a bank of questions, a quiz made from these questions, and a class response set) and then write queries on them.

Define DTDs

Below are the specifications for the three DTDs you will write. You must ensure that everything that is specified 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 real users would have a GUI that would do all the tedious parts; they should only have to enter the question text etc.

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

- There are three types of question: true-false, multiple choice, and numeric. Numeric questions will only take integer answers; we won't handle floats.
- Every question has a question ID, question text and a single correct answer, all of which are strings.
- A multiple choice question has answer options, and there are always at least two.
- For a multiple-choice question, any incorrect answer may have a hint associated with it.
- A numeric question may have one or more hints, each of which is associated with an incorrect answer in a given range. The range will be specified by its lower (inclusive) bound and its upper (non-inclusive) bound. For instance, if the range for a hint is (3, 7), it is a hint associate with an answer x such that $3 \leq x < 7$.

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

- A quiz has an ID, a title and one or more questions from the question bank.
- A quiz records whether or not students should be shown a hint (if there is one in the test bank) when they give a wrong answer to a question. This is a single flag for the whole quiz rather than one per question.
- Each question has a weight: an integer that indicates how much a correct answer contributes to the student's total score on the quiz.

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

- The set contains at least one student's responses to the quiz.
- A student's response also records the student's ID, which is a string.
- For each question on the quiz that the student answered, the student's response stores the answer the student gave. A student may not have answered all the questions.

Don't get distracted by wondering whether students can answer a question multiple times, when hints will be give, or other aspects of taking the quiz. Your only responsibility is to record the relevant data.

Create valid instance documents

Create a valid instance document for each of your three DTDs, using the data available on the Assignments page of the course website. Each of these 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 I am providing.

Name your files question-bank.xml, quiz.xml and class.xml. Run xmllint on your files to confirm that they are valid.

Write queries

Write queries in XQuery to produce the following results:

1. The text of every question in file bank.xml.
2. The answer of every multiple choice question in bank.xml.
3. The average question weight among the questions in quiz.xml.
4. The text of the question in quiz.xml with the highest weight.
5. For every student in class.xml and every question that they did not answer, the student ID and question ID.

6. An xml document (satisfying the DTD in file revolution.dtd) that contains the response and student ID for every student in class.xml who answered question N-15.
7. An xml document (satisfying the DTD in file summary.dtd) that contains the weight, question text, and correct answer for each question in quiz.xml.
8. An xml document called grades.xml (satisfying the DTD in file grades.dtd) that contains the student ID and total score of of every student in quiz.xml.

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

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.

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. There will again be some marks for aesthetics.

XQuery is very "fiddly". 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 your question bank and your quiz, 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"?

Capture your results

I will post on the Assignments page a script that runs xmllint on each of your xml files, and galax-run on each of your queries. It will send your output to a file called Part1-results.txt. If you are not able to complete a query, you may comment it out of the script.

What to hand in for Part 1

- Your three DTD files: question-bank.dtd, quiz.dtd, and class.dtd.
- Your three XML file: question-bank.xml, quiz.xml, and class.xml.
- Your query files, q1.xq through q8.xq.
- Your "capture" script (even if you didn't change it) and your results file, Part1-results.txt

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

Part 2: Database Theory

Answer the following questions. You must show your rough work or you will not receive credit for your answers.

1. Consider a relation R with attributes $ABCDEFGH$ and functional dependencies

$$F = \{A \to CD,\ C \to F,\ G \to BHD,\ CD \to G,\ BFG \to H,\ BCG \to FH,$$

$$DF \to E,\ DEF \to BCD,\ AD \to G,\ E \to ACD\}$$

   (a) Find a minimal basis for these functional dependencies. Put your FDs into alphabetical order.

   (b) Provide a decomposition of R that is in 3NF. Indicate the key(s) of each relation.

2. Consider a relation R with attributes $HIJK$ and functional dependencies

$$F = \{JK \to H, JK \to I, H \to J, I \to K\}$$

   Provide a BCNF decomposition of R. Project the dependencies onto each relation in your decomposition. Make sure it is clear which relations are in the final result.

What to hand in for Part 2

Your answers must be typed. Hand in a single file, either Part2.pdf or Part2.txt.