University of Toronto
CSC**343S, Winter 2013**

Assignment 3

Due: Wednesday 03 April, at 10:00 pm sharp!

IMPORTANT: Read the Piazza discussion board for any updates regarding this assignment. I will provide an FAQ there than summarizes what you need to know.

You can do Parts 1 and 2 of this assignment in any order.

Part 1: XML and DTD design

Introduction

This part of the assignment concerns data about rental properties, propery owners, and renters. You will design DTDs for XML files that can store the necessary data; create XML files that and are valid with respect to your DTDs; and write a few simple queries on your dataset.

Note that your files will only store data that represents a snapshot in time. They will not record any historical information, such as who used to rent a property or who the owner used to be.

Define DTDs

Below are the specifications for three DTDs you will write. You must ensure that any data that fits the description below can be expressed in an XML file that conforms to your DTD. There may be things that should be *dis*allowed but that you cannot enforce because of the limitations of DTDs. That's okay. Express as many of the restrictions as you can using the features of DTDs that I have shown you.

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 an 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.) Also, remember that sometimes information can be inferred; it doesn't always have to be explicitly stored.

Don't compromise what you consider to be good design in order to have XML files that would require less typing to create. 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.

In a file called **property.dtd**, specify the format for an XML file containing information about rental properies. It must support the following features:

- There are two types of property: commercial and residential.
- All properties have an address, which is unique and a single an owner.
- A commercial property has a number of square feet.
- A residential property has a number of bedrooms, whether or not it has central air conditioning, and whether or not utilities are included in the rent.
- A property is either single-unit or multi-unit. If it is a multi-unit properly, it has at least 2 units and each unit as an ID that is unique within the property, such as "apartment 2112" or "unit 6".
- For each single-unit property, and each unit in a multi-unit property:

  - It has a value for rent, which is the amount a person must pay to rent it. The rent has an associated period that the rent covers; for instance rent could be monthly or yearly.
  - It is either currently rented or not. If it is rented, one renter ID is recorded.

In a file called **owner.dtd**, specify the format for an XML file containing information about the owners of the rental properties. It must support the following features:

- An owner is either an individual or corporation. Regardless, the owner has a name, ID, and address.
- An owner has at least one phone number. If the owner is a corporation, each phone number has a name and a title associated with it (for example, Marissa Mayer, CEO).

In a file called **renter.dtd**, specify the format for an XML file containing information about renters and potential renters of properties. It must support the following features:

- A renter has a name and ID.
- A renter has zero or more phone numbers. Each phone number has a type, which is either "cell", "home", "work", or "other".
- A renter is renting zero or more properties, and the ID of each property they are renting is recorded.

These general facts are also relevant:

- An address consists of an optional unit number, a street number, street name, city, province, country and postal code.
- Phone numbers consist of a country code, area code, and local number.

Create valid instance documents

Create instance documents called `property.xml`, `owner.xml` and `renter.xml` that are valid with respect to your DTDs. Include at least 2 commercial properties, 2 residential properties, 2 single-unit and 2 multi-unit properties in `property.xml`. Include at least 2 owners and 2 renters in the other files.

Make each of these instance documents separate from its DTD; in other words, do not embed the DTD in the XML file for the instance document.

Run xmllint on your files to confirm that they are valid with respect to your DTDs.

Write a query

Write a query in XQuery to find, for each single-unit property with two bedrooms that is rented, the property ID, owner's name, and renter's name. Do not hard-code your query to match the data in your particular documents; your query should work on any valid instance documents. Put enough data in your xml files that the query will yield two results. Format of the output does not matter.

Store your query in a file called `part1.xq`.

Capture your results

I will post on the Assignments page a very simple script called `run-part1.sh` that runs xmllint on your XML files and galax-run on your query. It will send the output to a file called `part1-results.txt`.

What to hand in for Part 1

- Your DTD files: `property.dtd`, `owner.dtd` and `renter.dtd`.
- Your XML files: `property.xml`, `owner.xml` and `renter.xml`.
- Your query file: `part1.xq`.
- `part1-results.txt`.

Part 2: XQuery

In class, we have been using a set of XML and DTD files for a question bank, a quiz, and a set of class responses to the quiz. These might form part of the back end of an online system for giving quizzes.

For this part of the assignment, I will provide modified versions of these files that include two differences:

- For multiple-choice questions, OIDs are no longer included. We can simply index into the list of options to refer to a specific answer, as in `class.xml`.
- The QIDs are now simple numbers. The question bank already encodes question type, so it is not necessary to repeat that information within the QID value.

Your job will be to write queries on files that satisfy these DTDs. Your queries should of course return correct answers on the specific XML data files I provide. But they must, in fact, return correct answers on *any* XML data files that satisfy the DTDs. You should definitely test your queries on other datasets.

Here are a few things I'd like you to notice about the DTD and XML files:

- A student may not have answered all the questions on a quiz.
- A multiple-choice or numeric question may have hints associated with some or all of the incorrect answers. For numeric questions, the incorrect answers can't simply be enumerated. Instead, their hints are associated with incorrect answers in a given range, specified by its lower and upper bound.
- Since an instructor may not want to give hints, a quiz records whether or not this should happen. This is a single flag for the whole quiz rather than one per question. You may have questions about when hints are shown to students. But you are not implementing software to give quizzes, so all you need to know about hints is enough to understand what the queries below are supposed to produce.
- QID values from the question bank are recorded as QID values in the quiz and in the response set. Do the DTDs enforce these"foreign keys" across files?
- In the question bank, each multiple-choice question has an `answer` attribute, and its value is a number that identifies which answer option is correct. The number 1 refers to the first answer option, 2 refers to the second, and so on. Similarly, when the class response set has to record which response a student gave, it uses the same scheme.

Write queries

Write queries in XQuery to produce the results described below. For the queries that generate XML output, generate only the XML elements. Don't try to prepend that with the declaration information that belongs at the top of the file. For all other queries, format of the output does not matter. Just try to make your output readable.

These are not listed in order of difficulty. Tackle them in any order.

1. The QID of multiple choice questions with fewer than three answer options.
2. The text of all questions that have no hints.
3. For each question in the question bank, the content of the question text and the content of the correct answer.
4. The QID of questions from the question bank that are not on the quiz.
5. The QID of questions on the quiz that fewer than half the students answered, or that fewer than half of those who answered actually got right.
6. The SIDs of pairs of students whose answers on the quiz were exactly the same. In other words, the set of questions they gave answers to is the same, and the answers are the same also.
7. An XML document (satisfying the DTD in file distribution.dtd) that contains the number of students who got 0 answers correct on the quiz, 1 correct, and so on, up to the maximum possible.
8. An XML document called report.xml (satisfying the DTD in file report.dtd) that contains a full report for each student in class.xml. This includes the student ID's and, for each question on the quiz, the QID, the student's answer to the question (or empty text if they gave none), their mark for that question and the student's total score.

Store each query in a separate file, and call these q1.xq through q8.xq. Do not hard-code your queries to match the data in your particular documents. Your query should work on any valid instance documents.

Here are a few XQuery reminders that may help:

- Although we spent most of our time on FLWOR expressions, remember that there are other kinds of expression. We've studied `if` expressions, `some` expressions `every` expressions, and expressions formed with the set operators `union`, `intersect` and `except`. 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 two things, 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?"

Testing your queries

I will post on the Assignments page a script called `run-part2.sh` that validates your instance documents, runs each of your queries and prints the results, and then validates the XML for any queries that produce XML. (It will prepend the declaration part that your queries do not generate.) It will send your output to a file called part2-results.txt.

We will run your code through the same script, but using our own instance of the XML files. Just in case any problems arise, you will also hand in your own XML files and part2-result.txt file.

When doing your own testing, you may make a modified version of the script if you find that helpful. But make sure you hand in results from the original script. If you are not able to complete a query, you may comment it out. Because of this possibility, I will ask you to hand in the script as well.

What to hand in for Part 2

- Your XML files: `bank.xml`, `quiz.xml`, and `class.xml`.
- Your query files: `q1.xq` through `q8.xq`.
- Your `run-part2.sh` script (even if you didn't change it) and your results file, `part2-results.txt`

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

Part 3: Functional Dependencies, Decompositions, Normal Forms

This will be posted after we have covered the material in class.

Marking

The final marking scheme has not been set yet, however, you should expect that Part 2 will be worth the most. Parts 1 and 3 will be worth roughly 25% each, and Part 2 roughly 50%.

Some parting advice

It will be tempting to divide the assignment up with your partner. Remember that both of you probably want to answer all questions the final exam. `:-)`

There are a lot of files to hand in! Don't leave assignment submission to the last minute.