University of Toronto
CSC**343S, Fall 2014**

Assignment 3

Due date revised from course syllabus: **Mon 1 Dec**, *at 10:00 pm sharp!*

IMPORTANT: Read the Piazza discussion board for any updates regarding this assignment. We will provide a summary of key clarifications, and it is required reading. Check it regularly for updates.

Part 1: XQuery

For this assignment, our domain is an online job application system. We are providing DTDs (in files `posting.dtd`, `resume.dtd`, and `interview.dtd`) for files to store data on job postings, resumes, and interviews. Your job will be to write queries that will work on any files that satisfy these DTDs. Here are a few things I'd like you to notice about the DTD design:

- A resume's *summary* is the statement that people often put at the top, for instance to indicate their objective.
- An *honorific* is a title by which we refer to a person, such as Ms, Dr., or Professor.
- A *title* is a job title, such as "Chief Technology Officer".
- In a resume, the empty tag *honours* doesn't have any attributes. It would seem to be pointless, but its presence is used here to indicate that a degree was an honours degree.
- A job posting may include questions. These are questions of special importance to the position that the interviewer is encouraged to ask.
- Some values act as foreign keys across files. For example, the *rID* recorded for an interview is a reference to the *rID* of a resume, and these values are in different files. DTDs do not have the power to enforce the validity of these references, but you may assume that any XML files we run your queries on will not contain invalid references.

Create instance documents

Create a instance document for each of these DTDs, with any data in them that you like. Call them **resume.xml**, **posting.xml** and **interview.xml**. This will help you understand the DTDs and to picture what it is your queries will be acting upon (pretty essential for writing the queries!). You can also use these instance documents as part of your testing.

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. Be sure to validate your XML files against the DTDs.

Write queries

Write queries in XQuery to produce the results described below.

Except for queries 7 and 8, the structure of your output doesn't matter (whether it is elements, or the contents of elements, for example). This is because the evaluation of your output will not be automated; TAs will read your output instead. Just ensure that the necessary information is presented.

For the queries that generate XML output, generate only the XML content; don't try to prepend that with the declaration information that belongs at the top of an XML file.

For all queries, the whitespace in your output doesn't matter. Don't worry about formatting it attractively.

1. Find all resumes where more than 3 skills are listed. Report the rID, forename, and number of skills.

2. Find all interviewers who have given an interview and provided no assessment of collegiality. Report their sID. Do not include duplicates.

3. For each posting, find the required skill that has been given highest importance (in the case of ties, report all). Report what the skill is and its importance.

4. Find postings that have a required skill that does not occur at the required level (or higher) in any resume. Report the pID.

5. Find postings that have a required skill such that (a) fewer than half the resumes include that skill or (b) of the resumes that include the skill, fewer than half list it at a level above 3. (This is an inclusive or; postings that satisfy both (a) and (b) should be included.) Report the pID.

6. Among those resumes that list at least one skill, find pairs of resumes that list exactly the same skills at exactly the same level. For each pair, report the two rIDs.

7. Generate an XML document called `skills.xml` that reports, for each skill listed in any posting, the number of resumes that list that skill at level 1, at level 2, and so on. Your XML output must validate against the DTD in file `skills.dtd`.

8. Suppose we are interested in how well a person's skills match what a posting requires. Here is an example:

Posting			Resume	
what	level	importance	what	level
SQL	4	5	SQL	5
R	3	4	R	4
Scheme	3	3	Scheme	skill not listed
LaTeX	5	3	LaTeX	5
Python	4	4	Python	3

   Let's give the person n points for each skill they possess at the required level or higher, where n is the importance of that skill to the position. And similarly we will subtract n points for each skill they don't possess at the required level. In our example, the person's score would be $5 + 4 - 3 + 3 - 4 = 5$. Let's call this the person's "degree of match" for the posting.

   Generate an XML document called `report.xml` that contains a summary report for each interview. This includes the rID and name of the person interviewed, the position they were interviewed for, their degree of match for the position, and their average score on all elements of the assessment. Your XML output must validate against the DTD in file `report.dtd`.

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

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

We will post a script called `runall.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 send output to a file called `results.txt`. When doing your own testing, you can still use the script if you comment out any queries that you have not yet implemented successfully.

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 `result.txt` file. We will ask you to hand in the version of `runall.sh` that you used to generate these results as well, even if you have not modified it.

What to hand in for Part 1

- Your query files: `q1.xq` through `q8.xq`.
- Your XML files: `posting.xml`, `resume.xml`, and `interview.xml`.
- Your results file, `results.txt`.
- The `runall.sh` script that you used to generate those results (even if you didn't change it).

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

Part 2: Functional Dependencies, Decompositions, Normal Forms

1. Consider a relation schema R with attributes $ABCDEFGH$ with functional dependencies S:

$$S = \{A \to CF, \quad BCG \to D, \quad CF \to AH, \quad D \to B, \quad H \to DEG\}$$

   (a) Which of these functional dependencies violate BCNF?

   (b) Employ the BCNF decomposition algorithm to obtain a lossless decomposition of R into a collection of relations that are in BCNF. Make sure it is clear which relations are in the final decomposition and project the dependencies onto each relation in that final decomposition. Because there are choice points in the algorithm, there may be more than one correct answer. But you must follow the BCNF decomposition algorithm.

   Show all of your steps so that we can give part marks where appropriate. There are no marks for simply a correct answer.

2. Consider a relation R with attributes $ABCDEF$ and functional dependencies S:

$$S = \{AB \to EF, \quad B \to CEF, \quad BCD \to AF, \quad BCDE \to A, \quad BCE \to D, \quad DF \to C\}$$

   (a) Compute all keys for R.

   (b) Compute a minimal basis for S. In your final answer, put the FDs into alphabetical order.

   (c) Using the minimal basis from part (b), employ the 3NF synthesis algorithm to obtain a lossless and dependency-preserving decomposition of relation R into a collection of relations that are in 3NF.

   (d) Does your schema allow redundancy?

   Show all of your steps so that we can give part marks where appropriate. There are no marks for simply a correct answer.

What to hand in for Part 2

Hand in your typed answers, in a single file called Part2.pdf.

Marking

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

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.