University of Toronto
CSC**343S, Winter 2015**

Assignment 3

Due date extended from that published in the course syllabus: **Thu 2 Apr**, *at 8:00 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 music service such as Spotify, Google Play Music, or Deezer. We are providing DTDs (in files `music.dtd` and `users.dtd`) for files to store data on songs, playlists, users, and who follows whom. Your job will be to write queries that will work on any files that satisfy these DTDs.

Here are a few comments about the DTDs and the domain:

- A playlist is a list of songs (also called "tracks") that a user has created. Playlists are defined in a music file — a file that is an instance of the DTD defined in `music.dtd`.
- If a user has listened to a playlist, whether they or someone else created it, it is stored in their user profile with a count of how many times they've listened to it.
- Users can follow each other, as on facebook.
- Songs, playlists and users all have identifiers, stored as values of the attributes `sid`, `pid` and `uid` respectively. These are intended to serve as keys, hence they have type `ID` in the DTD file. Note that since song identifiers and playlist identifiers are defined as `ID` attributes in the same file, there can be no overlap in their values.
- These identifiers are also used as foreign keys. For instance, `follows` elements refer to a user by their `uid` value. However, references across files are not supported by DTDs. This is why the `creator` attribute of a `playlist` element is defined to be plain `CDATA` rather than `IDREF`, even though it refers to a `uid`. You may assume that all values for `creator` correctly refer to a `uid` in the users file.

Create instance documents

Create a instance document for each of these DTDs, with any data in them that you like. Call them **music.xml** and **users.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. Each query has an associated DTD file. These are called q1.dtd, q2.dtd etc. All of your queries will generate XML content and must be valid with respect to their DTD. Don't prepend your query result with the declaration information that belongs at the top of an XML file; we will provide a script that runs each query, prepends the XML declaration to your query output, and runs xmllint to validate it.

For all queries, the whitespace in your output doesn't matter. Don't worry about formatting it attractively. Throughout, include no duplicates unless the question says to include them.

The queries are not listed in order of difficulty. Tackle them in any order.

1. Find all songs that are on no playlist. For each, report the song id.

2. Find all users with fewer than 4 followers. For each, report their user id, as well as the user id of each of their followers. If they have no followers at all, report just their own user id.

3. For each user, find which of their playlists has the highest playcount. (In the case of ties, report all). For each, report the user id, playlist id, and playcount. If the user has no playlists, report just the uid.

4. Find pairs of users such that the two users have exactly the same set of playlists and that set has at least 5 playlists in it. For each pair, report the two user ids. Report each pair only once — don't repeat the information by reporting the pair in the reverse order. Also be careful not to report a user paired with him or herself.

5. For each playlist, report the number of users who have that playlist with a playcount less than 5, the number who have it with a playcount between 5 and 49 inclusive, and the number who have it with a playcount of 50 or higher.

6. A playlist has a playcount for every user that has that playlist. Let's call the sum of all these playcounts for a particular playlist its overall playcount. Let's say the playcount of a song is the overall playcount of every playlist that the song appears on. Find the playcount of every song. Report the song id, title, and playcount.

Store each query in a separate file, and call these q1.xq through q6.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 `q6.xq`.
- Your XML files: `music.xml`, `users.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 $ABCDEFGHI$ with functional dependencies S:

$$S = \{A \to BC, \quad AD \to E, \quad BD \to FG, \quad BDH \to I\}$$

   (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:

$$S = \{ACDE \to F, \quad B \to DF, \quad BCDF \to A, \quad BD \to CEF, \quad BEF \to CD\}$$

   (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.