University of Toronto
CSC**343, Fall 2015**

Assignment 3

Due date: **Wed 9 Dec**, *at 10:00 pm sharp!*

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

Part 1: XQuery

Our Miniature Internet Movie Database (MIMDb) contains semi-structured data that holds information about Hollywood movies, actors, directors and Oscar awards. We are ultimately interested in designing a website that enables users to browse this data and extract useful information. You will be given the document type definitions (DTDs) of all the main entities in the MIMDb schema. Your job is to implement a list of **XQueries** that will extract specific information from the schema.

Additionally, you will be given some XML files to help you test the correctness of your queries. However, your queries should be designed to return correct answers on *any* XML data files that satisfy these DTDs. It is recommended that you test your queries on other datasets besides the ones provided for you here. The main entities in the MIMDb schema and the DTDs that describe them are as follows:

- **Movies.** In the file `movies.dtd` you will find the specification for XML files that hold information about movies, such as the year they were produced, the actors who appeared in them, or the Oscar awards they won (if any).
- **People.** Specification about XML files that contain personal information about actors and directors can be found in `people.dtd`, including their names, gender, date-of-birth, or any Oscar awards they might have won, for example.
- **Oscars.** In `oscars.dtd` you will find the specifications for XML data related to Oscar awards that have been awarded either to movies or individuals (actors/directors), including the year of the award and its type (i.e. name).

Notice the following supported features in the MIMDb schema:

- A person can be both an actor and a director (whether in the same movie or in different movies).
- An Oscar award can be associated with a person only, a movie only, or both a person and a movie (e.g. for a person's role in a certain movie).
- A movie can be directed by one director only, but will have one or more actors (no less than one actor).

Query XML Documents

Write queries in XQuery to produce the following results:

1. [8 points] The average number of oscars won by any person (whether an actor or director).

2. [8 points] For every movie in the database, the movie ID and the number of actors who acted in it.

3. [8 points] Which directors do not have their place of birth (pob) listed in the database? Return their person ID (PID) and last name.

4. [8 points] All movies that were directed by James Cameron since the year 2001. Assume there is only one "Person" with the name James Cameron. Return the movie title and the movie year.

5. [12 points] For each Oscar award type ever given to a *movie*, find in which year it was awarded for the first time. Return the name of the Oscar award (Type), the year it was first awarded, and the title of the movie it was awarded to.

6. [12 points] Which actor(s) were also directors to one or more movies? Return their person ID (PID) and last names.

7. [12 points] A valid xml (satisfying the DTD in file `stats.dtd`) that contains a histogram showing the number of movies that appeared in each genre (main category).

Store each query in a separate file, and call these **q1.xq** through **q7.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.

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 *movies* and *people*, 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

Capture your results by running the script `runall.sh` found on the course website. It will run galax-run on each of your queries and sends the output to a file called results.txt.

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 1

You should submit your the following files electronically by committing them to your svn repository:

- Your query files: `q1.xq` through `q7.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.

Part 2: Functional Dependencies, Decompositions, Normal Forms

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

$$S = \{BH \to AD, \quad D \to BH, \quad BCE \to F, \quad F \to C, \quad A \to GEF\}$$

   (a) [2 points] Which of these functional dependencies violate BCNF?

   (b) [12 points] 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 $ABCDEFG$ and functional dependencies S:

   $S = \{DBE \to FC,\ CD \to AF,\ D \to AB,\ D \to G,\ BADE \to C,\ ABD \to E,\ D \to F,\ EF \to B\}$

   (a) [6 points] Compute all keys for R.

   (b) [6 points] Compute a minimal basis for S. In your final answer, put the FDs into alphabetical order.

   (c) [4 points] 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) [2 points] 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.

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.