University of Toronto
CSC343, Winter 2017

# Assignment 3

*Due: Wednesday, 5 April, 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.

## Learning Goals

By the end of this assignment you should have:

1. become aware of the tradeoffs that must be made when designing a schema in the relational model,
2. become very comfortable expressing a schema in SQL's Data Definition Language,
3. gained intuition for what DDL can express in comparison to what an XML DTD can express, and thought about the relative strengths of the relational model and the semi-structure model for representing a domain,
4. become fluent in XPath, and able to use some of the features of XQuery when needed, and
5. learned how to formally reason about functional dependencies.

## Introduction

For this assignment, our domain is online job search. We are providing DTDs (in files `posting.dtd`, `resume.dtd`, and `interview.dtd`) for files to store data on job postings, resumes, and interviews. 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 includes one or more required skills. In addition to specifying what the skill is, it specifies the level at which the skill should be possessed, and the importance of the skill to the job.
- 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

To get familiar with the domain and the DTDs, 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**. You can also use these instance documents as part of your testing for the queries you will write in Part 2.

Make each of these instance documents separate from its DTD; in other words, do not embed the DTD in the XML file. Be sure to validate your XML files against the DTDs.

# Part 1: Informal relational design

In class, we are in the middle of learning about functional dependencies and how they are used to design relational schemas in a principled fashion. After that, we will learn how to use Entity-Relationship diagrams to model a domain and come up with a draft schema which can be normalized according to those principles. By the end of term you will be ready to put all of this together, but in the meanwhile, it is instructive to go through the process of designing a schema informally.

Your task in Part 1 is to construct a relational schema for our domain, expressed in DDL. It must satisfy these properties:

- Any data that could be represented in XML files that conform to our DTDs can be represented in an instance of your relational database.
- All constraints expressed in the DTDs are enforced by your relational schema, unless that is impossible. At the top of your DTD file, include a comment labelled "Count not enforce" that lists these DTD constraints that could not be enforced.

As you know, there are many possible schemas that satisfy these properties. We aren't following a formal design process for Part 1, so instead follow these general principles when choosing among possible options:

- There may be additional constraints that make sense in the domain but were not enforced by the DTDs. Enforce these, if possible, in your DDL. Add a comment labelled "Additional constraint" to each one so that we can easily find them in your DDL.
- Avoid redundancy.
- Wherever an attribute cannot be null (according to the DTD), add a `NOT NULL` constraint.
- Avoid designing your schema in such a way that there are attributes that *can* be null.

You may find there is tension between some of these principles. Where that occurs, use your judgment to make a tradeoff.

## What to hand in for Part 1

Hand in a file called `schema.ddl` containing your schema, as well as a plain text file called `part1-demo.txt` that shows you starting postgreSQL, importing `schema.ddl`, and exiting posgreSQL. You must hand in this demo and the file must be a plain text file or you get zero for this part of the assignment. If you are unsure about this, please talk to me.

# Part 2: XPath / XQuery

Write queries in XPath / XQuery to produce the results described below. Your queries must work on any files that satisfy our three DTDs.

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; our autotesting will prepend the appropriate XML declaration to your query output before running xmllint to validate it.

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

XQuery is very "fiddley". It's easy to write a query that is very short, yet full of errors. These can be difficult to find and the syntax errors you'll get are not as helpful as you might wish. A good way to tackle the queries is to start incredibly small and build up your final answer in increments, testing each version along the way. For all but the last query, I have suggested intermediate steps to help you. 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?"

1. Find all postings that require the skill SQL at level 5.

   **Tips:** Here is one suggested sequence of steps for solving this query:

   (a) Write a simple path expression to find all postings.

   (b) Filter the postings to include just those that require SQL.

   (c) Now we need to restrict ourselves to level 5. The trick is that we need the level 5 to be specifically for SQL.

   (d) Wrap the results in a `dbjobs` element.

   Note that you can nest a "`[]`" filter inside another one. This may come in handy.

2. Find the one or more job postings that include a skill whose value to the job (its level times its importance) is the highest across all postings. There could be a tie for highest.

   **Tips:** Notice that level times importance can never exceed 25 due to the limits imposed in the DTD, but you still have to find the maximum that actually occurs in the file. Here is one suggested sequence of steps for solving this query:

   (a) Find all the products of level times importance.

   (b) Find the maximum of these.

   (c) Separately, find all postings and put any silly filter on it which you can replace later. Ultimately, you will be producing a filtered list of postings.

   (d) Change the filter to be anything about level times importance among the required skills in the posting.

   (e) Modify the filter to find the desired postings by comparint to the maximum that you learned how to compute earlier.

   (f) At this point, you are producing a sequence of the desired postings. Wrap it in a `postings` element.

   Everything except for the final step can be done with just XPath. You may find the `parent` axis helpful at some point.

3. Find all resumes on which more than 3 skills are listed. Report the rID, forename, number of skills, and citizenship.

   **Tips:** Here is one suggested sequence of steps for solving this query:

   (a) Start with a FLWOR expression in which you have just a `let` to assign the parsed document to a variable, and a `return` whose expression that will ultimately evaluate to what you want to report. For now, return some arbitrary value like 123.

   (b) Add a `for` to iterate over the resumes. Change the `return` to report anything at all about each resume.

   (c) Add a `where` to filter the resumes as required.

(d) Change the return to have the necessary XML, but use the constant 123 everywhere some content is required, such as for attribute `rID` and element `name`.

(e) One at a time, replace the 123s with the correct content.

(f) Wrap the whole expression in a `qualified` element.

4. For each interview, report the forename of the person interviewed, the ID of the interviewer, and the highest score they got on any of these items in their assessment: communication, enthusiasm, and collegiality. (If there is a tie, report them all.)

   **Tips:** Here is one suggested sequence of steps for solving this query:

   (a) This query will require you to refer to two files. Start with a FLWOR expression in which you have a `let` to assign each parsed document to a variable, and a `return` whose expression that will ultimately evaluate to what you want to report. For now, return some arbitrary value like 123.

   (b) You are going to have to report something for each interview. Add a `for` to your FLWOR expression that iterates over the interviews. Change the `return` to report anything at all about each interview.

   (c) Add a `let` to find, for each interview, the highest score on communication, enthusiasm, or collegiality.

   (d) Add a `let` to find, for each interview, the skill element with that highest score.

   (e) Change the return to report that skill element.

   (f) Change the return to wrap the `best` element around the skill element. For now, make the two ID values be constants.

   (g) Replace the constants with the correct value. For `resume`, you will have to refer to the other document to find the person's forename. Fortunately, you have already parsed it.

   (h) Wrap the whole result in a `bestskills` element.

5. For each skill listed in any posting, report the skill and the number of resumes that list that skill at level 1, at level 2, and so on.

   **Tips:** By now, you should be able to break this down. Here are some specific bits of XQuery that may be helpful:

   - If an expression returns a sequence that includes duplicates, you can remove them by putting the expression inside a call to function `distinct-values`.
   - Remember that you can nest expressions arbitrarily. For example, a `return` can include a whole FLWOR expression
   - XQuery has a counted loop. Example: `for $i in 1 to 100`

Store each query in a separate file, and call these q1.xq through q5.xq.

Here are a few general XQuery reminders:

- Remember all the kinds of expression: path expressions, FLWOR expressions, `if` expressions, `some` expressions, `every` expressions, and expressions formed with the set operators `union`, `intersect`, and `except`.
- XQuery is an expression language. Each query is an expression, and we can nest expressions arbitrarily.

## What to hand in for Part 2

Hand in your query files: `q1.xq` through `q5.xq`. You may work at home, but you must make sure that your code runs on the cdf machines.

# Part 3: Functional Dependencies, Decompositions, and Normal Forms

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

$$S = \{\ I \to DGF, \quad H \to CEA, \quad BI \to J, \quad B \to H, \quad CI \to K\ \}$$

   (a) State which of the given FDs violate BCNF.

   (b) Employ the BCNF decomposition algorithm to obtain a lossless and redundancy-preventing decomposition of relation R into a collection of relations that are in BCNF. Make sure it is clear which relations are in the final decomposition, and don't forget to 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. List the final relations in alphabetical order (order the attributes alphabetically within a relation, and order the relations alphabetically).

2. Consider a relation $P$ with attributes $ABCDEFGH$ and functional dependencies $T$.

$$T = \{\ ACDE \to B, \quad BF \to AD, \quad B \to CF, \quad CD \to AF, \quad ABF \to CDH\ \}$$

   (a) Compute a minimal basis for $T$. In your final answer, put the FDs into alphabetical order. Within a single FD, this means stating an FD as $XY \to A$, not as $YX \to A$. Also, list the FDs in alphabetical order ascending according to the left-hand side, then by the right-hand side. This means, $WX \to A$ comes before $WXZ \to A$ which comes before $WXZ \to B$.

   (b) Using your minimal basis from the last subquestion, compute all keys for $P$.

   (c) Employ the 3NF synthesis algorithm to obtain a lossless and dependency-preserving decomposition of relation P into a collection of relations that are in 3NF. Do not "over normalize", this means that you should combine all FDs with the same left-hand side into a single relation. If your schema includes one relation that is a subset of another, remove the smaller one.

   (d) Does your schema allow redundancy? Explain how you know that it does or does not.

Show all of your steps so that we can give part marks where appropriate. There are no marks for simply a correct answer. If you take any shortcuts, you must explain why they are justified.

## What to hand in for Part 3

Type your answers up using LaTeX or Word. Hand in your typed answers, in a single pdf file called part3.pdf.

# Final thoughts

**Submission:** Well before the due date, declare your team on MarkUs. It is impossible to do so during the grace period, even if you have grace points you are going to use. Also, check that you have submitted the correct version of your files by downloading it from MarkUs; new files will not be accepted after the due date.

**Marking:** The marking scheme will be approximately this: Part 1 30%, Part 2 40%, and Part 3 30%.

**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.