Deliverables

There are six deliverables: `ERModel.pdf`, `Schema.sql`, `VotesImporter.java`, `Senator.java`, `Vote.java`, and `VoteCast.java`. Please submit the files for grading via Moodle.

Pair Programming

You are required to work with your assigned partner on this assignment. You must observe the pair programming guidelines outlined in the course syllabus — failure to do so will be considered a violation of the Davidson Honor Code. *Collaboration across teams is prohibited and at no point should you be in possession of any work that was completed by a person other than you or your partner.*

1 Introduction

In this assignment, we will collect voting data related to the 115th Senate, 2nd session, and populate a database. This assignment will give you experience collecting and parsing XML data, creating a basic entity-relationship design, and creating/manipulating a database using SQL.

1.1 (5 pts) Required Reading

- ☐ Syllabus
- ☐ Textbook Chapters 1 (Introduction), 2 (Introduction to the Relational Model), 3 (Introduction to SQL).
- ☐ Textbook Chapter 7 (Database Design and the ER Model) up to (including) Sec. 7.5.5.

1.2 Material

Examine the page

`https://www.senate.gov/legislative/LIS/roll_call_lists/vote_menu_115_2.htm`

In this page, you see 274 links representing the issues voted by members of the 115th Senate, 2nd session. In the left, under the "Vote (Tally)" column, you find links to individual voting sessions. Each voting session has an associated XML description. All 274 voting session XML files have been downloaded and are provided in the `XML.zip` file[1]. If you are using Eclipse, you should extract the `XML.zip` file, and import the whole directory under your Eclipse project name (**not** under the "src" directory). If you are using VSCode, just put the directory at the same level as the code.

1.3 XML Structure and Semantics

Examine the file `vote274.xml`. The file represents a voting session (called simply `Vote`) in which multiple members (`Senator`s) *cast a vote* on a particular issue.

In the top-level hierarchy, there is information pertaining the vote. Under the unique `<members>` tag, you have multiple `<member>` tags, each describing the vote cast by a particular senator on this vote.

A **Senator** is defined as having a first name, a last name, party, and state. Each senator also has a unique identifier, noted in the tag `<lis_member_id>`. This identifier is consistent across all files.

A **Vote** is defined as having a congress number, session, year, date, and a number x. A vote with number x was the x-th vote on the appropriate $\{\text{congress number}, \text{session}\}$.

Each **VoteCast** represents the vote cast by a **Senator** in a specific **Vote**, along with his/her voting option: a *Yea*, a *Nay*, and, for our purposes, an *Absent* (which captures any other scenario, including the "Not Voting" indication found in the XML files).

It is part of your task to understand the structure of these files.

2 (20 pts) Entity-Relationship Model

Deliverable 1. Create an entity-relationship model for the setting above. Your diagram **must indicate** the primary keys of each entity. In the relationships, you **must indicate** the extra attributes of the relationship. You should also indicate the multiplicity of the

[1] Curious about how those files have been downloaded? See the `download.py` script along the XML files.

relationships, and whether they are total/partial **using the notation we adopted in class**. Deliver the model via Moodle under the name `ERModel.pdf`.

3 (20 pts) Database Schema

Deliverable 2. Create a file called `Schema.sql` that performs the following:

1. Creates a schema called "SenatorVotes", and makes it the default schema.
2. Creates all the relations representing the **two** entities and **one** relationship described in Sec. 1.3.
   (a) Name your relations and attributes consistently.
   (b) Carefully identify the domain for each attribute, as well as the number of bytes allocated for representation.
   (c) Indicate **all** the primary keys and foreign keys in each relation.

4 (50 pts) Populating the Database

Deliverables 3-6. Complete the skeleton program provided as part of this assignment, so that it parses all the 274 XML files under the "XML" directory and generates a SQL script that inserts exactly 102 Senators (there were substitutions), 274 Votes, and 27391 vote cast tuples into the database you created above.

If you are using Eclipse, create a new Eclipse project and import (i.e. drag) the source files into the "src" directory. You should then extract the ZIP file, and import (i.e. drag) the whole directory under your Eclipse project name (**not** in the "src" directory). If you are using VSCode, simply extract the file in the same level as the code. You should complete the `VotesImporter.java`, and modify the files `Senator.java`, `Vote.java`, and `VoteCast.java`. **You should not create any other files or classes.**

1. Look first at the `Senator.java`, `Vote.java`, and `VoteCast.java` files. Examine the `equals()` and `hashCode()` methods in the first two files. Try to map all those files to the relations you identified when you created your database schema.
2. In `VotesImporter.java` you need to complete `parseXmlFile()` and `generateSQL()`. You will have to create class-level attributes to maintain global information collected

from different XML files.

Attention. The absolutely most crucial design choice for your `VotesImporter` is what kind of data structures you choose to create as attributes of the class. This choice should be guided by the following **requirements**:

- The Senator information (ID, name, party, ...) is duplicated across different votes. Your script must have **exactly** 102 SQL statements to insert Senators into the appropriate table[2], **exactly** 274 SQL statements to insert Votes into the appropriate table, and **exactly** 27391 SQL statements to insert each vote cast into the appropriate table[3], **one statement per line, separated by semicolons**.
- You **must not parse** the names, party, and state of a Senator that has **already been identified** in a previous file. In each vote cast, you should first identify the Senator's ID, and then check if there's already a Senator associated to it.

5 (5 pts) Extra Credit

I expected to find 27400 votes instead of 27391 votes. You get 5 extra points if you either (a) find which 8 votes have not been assigned through the sessions; or (b) show that you have indeed 27400 votes and that my own accounting is incorrect. Make sure to tell me how you found that information (through your program, of course)!

6 (5 pts) Style

The remaining 5 points are granted for meeting the following style guidelines. The ER-diagram is tidily organized. The SQL script is well-indented and organized. Every new method you create should have a Javadoc. Variables and methods should be named consistently, and follow the standard Java coding conventions. The code is well-indented and organized (use Cmd-A Cmd-I in Eclipse, Cmd-P → Format in VSCode) to auto-indent your code). There are no absolute file references (so no "/Users/elmo/College/Spring19/..." anywhere in your code).

[2] You may have more or less than 100 because of substitutions.

[3] Can you find the 9 missing votes?

Good luck,
- Hammurabi