CS 327E Lab 2: Relational Database Design

Learning Objectives:

1. Continue working with the same real dataset(s)
2. Continue working with your partner
3. Continue working with the database schema that you created as part of Lab 1 and revise if necessary
4. Populate the database tables using SQL statements and Python
5. Gain some experience using a database connector

Prerequisites:

1. Lab 1 completed
2. Python 2.7 installed
3. Pip2 installed
3. Latest PyMySQL connector installed
4. Sublime Text editor installed
(see setup doc for how to install Python, pip2, PyMySQL and link to Sublime Text)

Steps Outlined:

0. Perform any reworks and/or enhancements to your Lab 1 submission that were noted in your graded rubric. For example, if we asked you to redefine an entity, you'll need to have made the necessary changes to the diagrams, data dictionary, and create tables script before continuing with this Lab.

1. Create a new folder in your local git repository called **lab2**. All the work you do for this lab will go into this folder. For now, copy all the datasets that you are working with into this folder. Commit this folder to your team's private repo and push the commit to your remote repo on Github.

2. Write an import script (separate Python program) for each table in your schema. The form of the import script will vary based on the type of table that is being populated. For base tables, the script reads in the raw data file(s) using the appropriate Python library (e.g. `csv` or `json` or `xml.etree`), does some re-formatting of the input data in Python as needed, and writes out the output to a database table . The output is written using `INSERT` statements executed through the PyMySQL connector (see lecture notes and code snippets for examples of how to do this). For junction tables, the script reads in the records from at least two base tables and writes out the output to the junction table. It may also read in one or more raw data file(s) if the many-to-many relationship contains attributes of its own.

Every import script reads the files it needs from a local directory and populates a different table. Name each script `import_<Table>.py`, replacing `<Table>` with the actual table name that the script is populating. For example, `import_Country.py` if the table name is `Country`. Make sure that each script contains some basic exception handling that prints out the error and the place in the code where the error occurred (again, see slides/snippets for examples of exception handling in this context). Commit the import scripts to your team's private repo and push the commit to your remote repo on Github.

3. Run each import script in Sublime using the Build option from the Tools menu and make note of the run time. The run time will appear in the output window below the executed code as: [Finished in x seconds]. In addition to the run time, make note of the total number of records imported into the table. Also, note any errors that occur during the import. Investigate the root cause of each error. If the error can be avoided by altering the code **without altering the raw data**, do so and re-run the script until the script is error free. Otherwise, make note of the error and provide a brief description of what is causing the error to occur during the import. An example of an error that is fixable is "Cannot add or update a child row: a foreign key constraint fails" as opposed to "Duplicate entry for 'mna34' for key 'PRIMARY'" which is not fixable without altering the raw data. Name each file `import_<Table>.txt`, replacing `<Table>` with the actual table name. Note that each file should contain the run time information, the number of records imported, and any import errors that could not be fixed along with their descriptions. You do not need to mention the fixable errors that you encountered. Commit the import txt files to your team's private repo and push the commit to your remote repo on Github.

4. Write a rollback script in Python for each table. The rollback script deletes all records from the table. It also resets the AUTO_INCREMENT on the table when one exists. This allows the import to be run multiple times without manual intervention between the runs. To reset the AUTO_INCREMENT, use the command:

```
ALTER TABLE <Table> AUTO_INCREMENT = 1;
```

The rollback script doesn't drop and re-create the tables. The rollback script assumes that each table already exists. Name the rollback scripts `rollback_<Table>.py`, replacing `<Table>` with the actual table name. Commit the rollback scripts to your team's private repo and push the commit to your remote repo on Github.

5. Write a main script that executes all the rollback and import table scripts. The main script should first call the rollback scripts followed by the import scripts. Ensure that the scripts are called in the right sequence, such that each parent table is populated before its children. The main script should print the return status of each script that it runs and report if the script ran successfully or with errors. If the error was caused by running the import in the wrong order, fix the code and re-run. Name the main script `populate_database.py`. Commit the main script to your team's private repo and push the commit to your remote repo on Github.

6. Ensure that all the scripts and datasets are in your lab2 folder on Github. Locate the last **commit id** that you are using for your submission and paste it into an email. Your email should also contain a link to your team's repo on Github. Address the email to the professor and both TAs and carbon copy your lab

partner. The subject of the email should be: [CS327E][Lab2][<TeamName>], replacing <TeamName> with your actual team name. The email is **due Tuesday, 10/18 at 11:59pm**. If it's late, there will be a **10% grade reduction per late day**. This late policy is also documented in the syllabus. Note: only one person per team should send the submission email.

Coding Conventions:

1. Place all reusable code in functions. For example, the database connection code should be defined in its own function because it's used throughout the program. Points will be deducted for duplicate code.

2. Use basic error handling with `try-except` blocks. Catch only the errors that you can handle and exit the program when an error is fatal (e.g. missing input data file, invalid database connection, etc). Use `print` statements in the `except` block to report the error regardless of whether you choose to continue or exit the program.

Teamwork & Collaboration:

1. We will use 3 class meetings (10/10, 10/12, and 10/12) to work on this lab.
2. We expect each team to split up the work as evenly as possible and both do multiple GitHub commits, open and resolve issues.
3. We expect each team to be making commits throughout the project. We don't want to see a single commit right before the assignment is turned in.
4. We expect each team to use the Github Issue Tracker to assign tasks and track their status. We will be reviewing the Issue Tracker to ensure that each team member is contributing to the project.

Resources:

Lab 2 Setup Document: http://tinyurl.com/hymam9a
Lab 2 Grading Rubric: http://tinyurl.com/haagutb
Lab 2 Team Sign-up Sheet: http://tinyurl.com/j6hzgvw
Code Samples: https://github.com/cs327e-fall2016/snippets