Three problems this week, though the first problem is very chill. The first two problems will deal with remote connections and transferring files back and forth between your local machine and a remote machine, whereas the last will work with establishing constraints for several SQL tables.

For the first two problems that work with the remote server, a reminder that the server is located at `165.232.142.111` and your username is the same as the start of your Willamette email. You should have already updated your password, so if you can't remember now what you updated it to (and you didn't get SSH keys setup), contact me and I'll see about resetting it for you.

For the third problem, in the repository is a template to serve as a starting point for creating the necessary tables. I've already included some queries at the top to create a new schema folder for this assignment and remove any tables that might be already existing. You can augment this starting point with the necessary column and constraint information, before checking your work against the other files as described in the problem.

In order to accept the assignment and get access to the repository, you should follow the link here:

Assignment link: https://classroom.github.com/a/XhYxs8sQ

1. As discussed in class, many shell programs allow you to customize them through the editing of a simple text file. This holds true for BASH as well! If you log onto the same server you accessed in class at `165.232.142.111`, you might notice that there is already a `.bashrc` file sitting in your home directory (you'll need to look for hidden files). If you open this file in Nano (`nano .bashrc`), you'll notice that there is already a lot of standard configurations placed in here, most of which are well documented with comments. Every time you open a new BASH shell, the contents in the `.bashrc` are read and executed. So if you want to have certain aliases, environment variables, or other customizations always available whenever you open a shell, adding them to the `.bashrc` is how you do so!

   One fun thing that can be edited in the `.bashrc` is the actual prompt that you see before your cursor. This information in stored in the variable `PS1`, and you can actually see it set on lines 60 and 62. The exact syntax and options for configuring this can be messy (especially if you want fun colors), but thankfully folks have gone to the effort of building an online bash prompt generator here. Create a custom prompt of your choosing using the available building blocks in the generator. Clicking on an element once you'd added it will also set you adjust its color and other attributes. Create something that you like and then copy the necessary command to *the end of the* `.bashrc` *document* (this will just ensure it overrides anything defined earlier in the file). Include a copy of your chosen prompt in the provided `Prob1.md` template file to submit this.

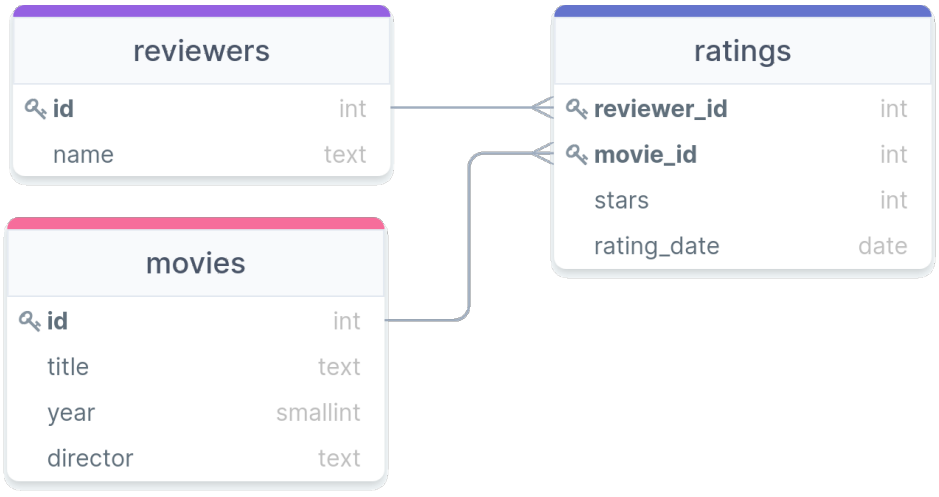
   If you want to do the same on your local system and you are using MacOS and thus are using ZSH instead of BASH, the relevant file is (unsurprisingly) `.zshrc` and there is an online prompt building here.

2. In other circumstances, servers might hold particular software or have greater resources available to them, and thus you want to run some program on the server. In many cases, you'll have local data on your own system that you want to upload to the server, run a program, and then bring the results back to your local system. On the same remote server you accessed in class (and in the above problem), there is a program installed called `word_finder` which accepts a single filename as an argument, where that file should contain a square, compact block of letters that comprise a classic word search. You can look at the contents of `puzzle_small.txt` in the repository as an example. When run with a given text file, this program will do 3 things:

   - output to standard output a table of all the words found within the puzzle and their corresponding definitions
   - save to the same location as the text file an image of the word search
   - save to the same location as the text file an image of the solution to the word search

   Your task here is to generate and save the results (all three outputs, *including* the word list) for the two example word searches included in the repository, as well as one of your own devising. Your own word search should have at least 3 words within it which are longer than 3 letters. To achieve this task, you will need to upload files to the server, run the program, and then bring the output back to your local system so that you can upload the results back to GitHub (**So 9 files to upload to GitHub!**). Given that you will need to be transferring several files, you *could* do them individually, but you could also use `rsync` to transfer all the contents in a particular folder (for example).

3. You'd like to set up some tables to keep track of some movies scores from a variety of reviewers. As such, you've sketched out a basic design shown in the below schematic:



   The repository file `Prob3.sql` has the initial layout of these tables, in addition to defining a schema to place them into. Using this file as your starting point, you should add the

necessary lines to create the correct columns and types for each table, including primary and foreign keys. Additionally, there are some further constraints that you would like to place on the data:

1. `(title,year)` in the `movies` table should be unique
2. `reviewers.name` should not be null
3. `ratings.stars` may not be null
4. `ratings.rating_date` may not be null
5. `movies.year` must be after the start of 1900
6. `ratings.stars` can only take on values of 1, 2, 3, 4 or 5
7. `ratings.rating_date` must come after the start of 2000

After adding your columns and constraints, you should be able to execute everything within `Prob3.sql` to create your necessary tables. To test that everything has gone well, you should then be able to run all the commands in `Prob3_Successes.sql` successfully, which should add rows to all of your tables without any errors. If an error occurs here, go back to your tables and determine what might have happened. Make sure your column names and types are just as indicated in the schematic.

Once all the commands in `Prob3_Successes.sql` complete correctly, trying to run *any* command in `Prob3_Failures.md` should result in an error owing to a constraint conflict of some sort. For each query (A-Q) in `Prob3_Failures.md`, indicate specifically what constraint caused the operation to fail. Use plain English here to describe the constraint, don't just regurgitate the error report that pops up! And realize that some queries will fail because of the same constraint. If any of these queries completes successfully, then you've missed a constraint somewhere, so go back to your table creation and review the schematic and above constraints to see what you might have missed.

Ensure that you upload both `Prob3.sql` and `Prob3_Failures.md` back to GitHub!

These ratings were all artificially generated for these mysterious reviewers, so if your favorite movie on the list didn't get an appropriate score, then I apologize!