Chris Helms

CSI 3335

Dr. David Lin

Programming Assignment

11/1/2019

This program was built using Python 3.7.4 & uses the cmd module of python. MySQL 8.0 was used for the database system & connects to Python 3.7.4 using the mysql.connector module. The Python datetime module was used to store times when used in the python program.

The only class of the project is Program which builds on the cmd module to run as a command line interface. This module includes help functionality as well as the ability to easily add user commands by defining functions with the name “do\_[NAME]” within the class. The user commands I created are:

#   do\_input\_files - used to populate the database

#   do\_route\_check - used to check for routes to/from 2 cities

#   do\_driver\_info - used to pull info on a driver

#   do\_city\_check - used to check all departures/arrivals to/from a city

#   do\_quit - quits program

I created a few additional helper functions to aid in the programming:

#   precheck - performs all prechecks on tuples before insertion

#   postcheck - performs all postchecks on tables after all insertions

#   translate - translates start depart day + a travel time into the arrive day

#   travelDays - determines all days of week a route travels on (uses translate)

The help for each user command is thorough. Since the purpose of this project was to work with a database, I did not spend a lot of time on user interface. Inputs must be entered in a specific way & are clearly outlined in the program introduction (prompt displayed when the program is started) & the help for each function. The program assumes proper user input & only the most obvious user input mistakes are corrected.

The file input function (do\_input\_files) reads 4 text file names without extensions to be read from the working directory. It enters data for routes, time\_table, bus\_driver, & driver\_assignments IN THAT ORDER (i.e. enter the text file names in that order). For each tuple, it first prechecks the tuple (if necessary) by running the precheck function. It then tries to prepare the tuple for insertion. After it has entered all tuples for the current table, it runs the postcheck function for all the tuples it has entered. It then displays how many tuples were successfully prepared for insertion. This is done for all 4 tables. After all the tables have been prepared, it will try to commit to the database. Warnings & errors are displayed to the user. Warnings involve a skipped tuple or that a non-vital condition has been violated. Errors are associated with a complete function failure.

The route check (do\_route\_check) function takes two cities entered as city, state, city, state (read help for this function for exact input) & finds the routes between them. It first finds all departures from the first city, state. It then finds all departures from the cities the departures arrived at. Next, it checks if any of the departures from the first city, state arrived at the destination city, state. This is recorded as a direct route. Next, for each departure from the first city, state, it checks if any of the departures from the arrival cities of that departure will arrive in the destination. If so, it checks if the layover time is acceptable. If so, it will record this as a transfer. Finally, all the direct & transfer routes are displayed to the user.

The driver info function (do\_driver\_info) takes as an argument first the type of search, then the search parameter. Read the help for this function for exact user input. It first searches for the driver by either name or ID as specified by the user. If there is more than one match, it will grab all of them. Next, for each driver it found, it will find all their assignments. Finally, it displays to the user each driver with their info & assignments.

The city check function (do\_city\_check) lists all departures & arrivals to or from a given city. The input is a city & state. Check the help for this function for exact user input. It first finds all departures from the city & the all the arrivals into the city. Next, it iterates though each arrival into the city & checks if it arrives on the day specified by adding the travel time. If it does, it adds it to the list of arrivals to present to the user. Next, it sorts all the departures by departure time & the arrivals by arrival time. It then displays both sets to the user.

The precheck function performs all the prechecks for the constraints specified. It takes as an argument an index corresponding to a table. Read through the program file for more details on how this function is used, it is explained in the comments. For the time\_table table, it checks if the route is a weekday only route & the new tuple is claiming it will run on a weekend or does not run during the week. It lets the input file function handle the error if the route does not exist. For the driver\_assignments table, it first cehcks if the day of week the tuple claims to run is an allowed day according to the time\_table entry it is associated with (i.e. does not try to run a route on a weekend if it is a weekday only route). Lastly, it simultaneously checks if the route will overlap an existing assignment, if there is enough rest, & if the driver can get to the next location. It does this by first parsing the new tuple entry. Next, it finds all the assignments for the driver associated with that tuple. Next, for each existing assignment, it checks if the new assignment overlaps. If it does not, it checks if it allows enough rest either before or after the new assignment. If it does, it checks if the driver will be left in the city the next assignment leaves from. If it does not, it checks ONLY IF there is a route that COULD take him to the next city. It does not check travel time, since it does not check when the route leaves or even if there is an assignment for one. If any of these checks fail, a flag is set to false & an associated warning message is prepared. If any check fails, the tuple will be skipped & a warning message is displayed to the user saying the tuple will be skipped & why.

The postcheck function performs all the postchecks for the constraints specified. It takes as an argument an index corresponding to a table. Read through the program file for more details on how this function is used, it is explained in the comments. For the time\_table table, it checks each route if it is NOT a weekday only route &, if it is NOT weekday only, it checks that there are time\_table entries to run on both the weekday & weekend. For the driver\_assignments table, it checks if there is an assignment for every day of the week that the route is supposed to run (i.e. if weekday only, it has an assignment for MWTRF &, if not weekday only, also on SU). If any of these checks fails, it simply displays a warning to the user that it has failed. It does not alter the table.

The translate function is just used to translate the departure time plus the travel time into the DAY that the assignment will arrive (not the time). It is used by multiple functions. The travelDays function is used for the last checks of the precheck function. It is used to create a list of days an assignment is traveling on. For example, if a route leaves Saturday & travels for 4 days (including Saturday), it will list as travel days Saturday, Sunday, Monday, & Tuesday. It does not return any times, just days.

A final note about this project. I am convinced you could find some weird edge case to break my program, but if the point of this project is truly to just give some experience in working with a database, I have proven my ability to do that. I reached a point of diminishing returns on thinking up & accommodating for edge cases especially given the purpose of this programming assignment.