CS457 Project 3

Yifeng Qin

11/05/20

1 How does my program organize multiple databases (Project 1)

The design of my program follows the example in the Assignment Overview.
One Directory corresponds to a database.
For example the parent directory will be /your_home/cs457
If you were to create a new database, it would create a new directory inside the parent directory.
Example: /your_home/cs457/db_1 (is what it would look like after creating db_1)
Then if you want to remove this database it would remove that directory with the db name.
Example: /your_home/cs457 (is what it would look like after removing db_1)
You can create as many distinct databases within the parent directory because they will all be represented
by a directory. This means you can also delete a database by deleting the corresponding directory.

2 How does my program organize multiple tables (Project 1)

One file corresponds a table in a directory.
For example the parent directory will be /your_home/cs457
To create a table you have to already created a database because you need to have a directory to place the
table file in.
You also have to USE a database to create a table in that database.
When wanting to create a table we join the parent directory with database directory and create a file with
the table name.
So each database can have the same distinct names.

3 How does my program store data (Project 2)

So all data that is inserted into a table is written to the corresponding file for that table. Meaning that
everything is written to the file as a string including numbers. Each line is separated by a newline and each
column is separated by a '|'. I created a helper function that reads all the data from the file and returns
the all the data in a list format. All the data that is read from the file is still in string format. Then I take
each line and make it it's own list making the data structure a list of lists. Each list represents a line and
each index in that line represents a specific column. I decided against a tuple because specific operations like
'==' and '!=' still compare numbers the same in string and number format. If I see a '>' or '<' operation
that means I have to compare numbers, meaning that I will cast the string to a float to perform the logic
operation.

4 How does my program join tables (Project 3)

To join the tables together we use a double for loop depending on the type of join. Since the the the '=' operator
is the same as the inner join operation. We can combine these two into one function meaning we only need

one function be called if we see these two. For this inner join we must find the index of both parameters we we checking and run through every combination to check if they match. If they match we can concatenate them together and add it to the list to print to the terminal. The outer join is very similar to inner join. Since it's a left outer join for this project we need all show all the data on the left side. But we must still match all the data on the right side. So we set a flag to check if we find a match and if we don't see a match we have to append only the left side to the list to be printed.

5   Functional Requirements

List the functional requirements that were listed in the assignment. Each functionality correlates to a function, which can be found at the end of the document.

5.1   Project 1

Database Creation - 4.2.1 create_database(self, db):
Joins the parent directory with the entered database name. Checks if the database already exists. If it exists already it will print out and error, if not it will create a directory of the database name.

Deletion - 4.2.2 drop_database(self, db) :
Joins the parent directory with the entered database name. Checks if the database already exists. If it does not exist it will print out and error, else it will delete a directory of the database name.

Table Creation - 4.2.4 create_table(self, tbl, inp) :
Gets the path to that table and will check if that already exists. Then if it does not exist, it will change the directory and create a file of that table name in that directory. If there are inputs to within the command it will write those variables to that file.

Deletion - 4.2.7 drop_table(self, tbl):
Joins the parent directory, database directory, and the name of the table, and if it exists delete that path. If it does not exist it will error and print the error message.

Update - 4.2.6 alter_table(self, tbl, inp):
Joins the parent directory, database directory, and the name of the table, and if it exists open the file and append the update values to file. If the table does not exist it will print and error message.

Query - 4.2.5 select_all(self, table):
Joins the parent directory, database directory, and the name of the table, and if it exits it will open the file and read all the contents into a list. Then it will print out all the contents to the terminal. If it does not exits it will print and error out to the terminal.

5.2   Project 2

Insert - 5.2.8 insert_table(self, table, new_data):
Checks if there the table is a valid path, then it will read all the current data from that table and store it in a list of lists. We take the new data that needs to be inserted and place them in the respective columns. We append that new data to the new current data. Then then call a helper function to write the new data back to the file.

Delete - 5.2.12 delete_items(self, table, data):
Checks if the table is a valid path. Reads all the data from the current table and stores it in a list of list.

Checks the logic operation to determine how to compare the values that need to be deleted. Then calls a helper function to delete the data from the current list of lists. This helper function will find the index of the variable we are comparing. Then we use the eval function to turn the string operation into a logic operation and perform it on the index of the variable we are comparing. If the variable we are comparing satisfies the condition we will delete that line from the current data. When all lines have been compared, it will call the insert helper function to write the new data back to the file.

Modify: - 5.2.10 update_table(self, table, data)
Checks if the table is valid. Then it will read all the data from the current table and store it in a list of lists. It will then pass all that information the update helper function to update the current data. This helper function will first find the index of the variable we want to compare and the variable we want to change. Uses the eval function to perform a logic operation based on the operation in the command. It will compare the index of the variable compared and if it satisfies the condition it will change the value at the index of the desired variable. Then it will call the insert helper function to write all the data back the file.

Query - 5.2.14 select_specific(self, var, table, where):
Checks if the table is valid. Then it will read all the data from the current table and store it in a list of lists. It find all the missing variables that are not needed to be shown. It will take all those unnecessary indexes and delete the columns from the print. Then it will call the delete helper function to get rid of rows that don't satisfy the where clause. This will return the final result to be printed to the terminal.

5.3   Project 3

Select - 4.2.5 select_all(self, table):
This function will determine how many different tables it has to show and determine if it needs to join the tables together. It will take the input command and then slice it to get the table names and if there is a join. We store the table names and then create a dictionary where the key is the name of the table and the value represents the data in that table in list form. Then it calls a helper function to determine if it needs to join the tables and calls the corresponding function to join it.

Inner Join - 6.2.20 inner_join(rows, res, d, tbls_alias, cols_id):
Will take the data from the two tables and run a double for loop to compare all the combinations. If will first look at the variables in each table and determine which columns need to be coma pared. Then it will look through all the combinations and if the columns match based on the logic operation it will concatenate them together and append it to the list to be printed. Because the inner join and the equals operator have the same implementation we just call this same function for both.

Outer Join - 6.2.21 outer_join(rows, res, d, tbls_alias, cols_id):
Will take the data from the two tables and run a double for loop to compare all the combinations. If will first look at the variables in each table and determine which columns need to be coma pared. Then it will look through all the combinations and if the columns match based on the logic operation it will concatenate them together and append it to the list to be printed. For this left outer join we have a flag variable in the first for loop to check if there is a match. If there is no match then we still need to include the data in the left side so we must append the data without a match from the right side.

6   Functions

This section contains all the code that runs the functionality of the program. Broken down into python files and functions.

6.1 main.py

Main driver that takes in the input from the file and calls functions in the run_script.py file to execute those commands.

6.1.1 read_file()

```
"""
Opens the file with standard input, and reads every line and checks if it is a valid
command to be
appended to the list. If there is a command on multiple lines then it will store it in a
 temp variable to be appended when it finds
the semicolon.
:return: Returns a List of the Commands
"""
commands = []
temp = []
for line in sys.stdin:
    if line == '\r\n' or line == '\n' or line[0:2] == '--':  # checks if it is a
relevant line to read
        continue
    else:
        if ';' not in line:
            temp.append(line.rstrip()+' ')
        else:
            temp = "".join(temp)
            line = re.sub(r"[\n\t]*", "", line)  # removes random special characters
like tabs
            commands.append(temp + line.rstrip())  # removes newline from line to append
 to list
            temp = []
return commands
```

6.1.2 run_commands()

```
"""
Calls the helper function read_file to get all the valid commands from the file. It will
 run through the list and
splice the commands to get rid of special characters.
:return: None
"""
commands = read_file()  # calls helper function to initialize to list commands
for command in commands:
    l = command.split(' ')  # splits the string command into a list based on spaces
    command = command.upper()  # converts the command to all uppercase so it can cover
case sensitivity
    size = len(l)  # gets length to handle missing spaces
    if 'CREATE DATABASE' in command:
        if size == 3:  # checks if all arguments are present
            script.create_database(l[2][:-1].upper())  # only gets the database name and
 removes the ';' from the back
        else:
            print('Syntax Error:', command)  # if size does not match there has to be a
syntax error with cmd
    elif 'DROP DATABASE' in command:
        if size == 3:  # checks if all arguments are present
            script.drop_database(l[2][:-1].upper())   # only gets the database name and
removes the ';' from the back
        else:
            print('Syntax Error:', command)  # if size does not match there has to be a
syntax error with cmd
    elif 'DROP TABLE' in command:
        if size == 3:  # checks if all arguments are present
            script.drop_table(l[2][:-1].upper())    # only gets the database name and
removes the ';' from the back
```

```python
                else:
                    print('Syntax Error:', command)  # if size does not match there has to be a
    syntax error with cmd
        elif 'USE' in command:
            if size == 2:  # checks if all arguments are present
                script.use_database(l[1][:-1].upper())  # only gets the database name and
    removes the ';' from the back
            else:
                print('Syntax Error:', command)  # if size does not match there has to be a
    syntax error with cmd
        elif 'CREATE TABLE' in command:
            if size >= 3:  # checks the the minimum amount of arguments are present
                command = " ".join(l[3:])  # gets all the variables after the table name and
     converts it into a string
                command = command[1:-2]  # then slice off the beginning '(' and the ');' at
    the end
                script.create_table(l[2].upper(), command)  # passes in the name of the
    table and the sliced variables to input
            else:
                print('Syntax Error:', command)  # if size does not match there has to be a
    syntax error with cmd
        elif 'SELECT * FROM' in command:
            if size == 4:  # checks if all arguments are present
                script.select_all(l[3][:-1].upper()) # only gets the table name and removes
    the ';' from the back
            else:
                print('Syntax Error:', command)  # if size does not match there has to be a
    syntax error with cmd
        elif 'ALTER TABLE' in command:
            if size >= 4:  # checks if all arguments are present
                command = " ".join(l[4:])  # gets all the variables after the table name and
     converts it into a string
                command = command[:-1]  # removes the ';' from the back of string
                script.alter_table(l[2].upper(), command)  # passes in the name of table and
     sting of variables
            else:
                print('Syntax Error:', command)  # if size does not match there has to be a
    syntax error with cmd
        elif 'INSERT' in command:
            if size >= 5:  # checks if all arguments are present
                script.insert_table(l[2].upper(), l[3:])
            else:
                print('Syntax Error:', command)  # if size does not match there has to be a
    syntax error with cmd
        elif 'UPDATE' in command:
            if size >= 8:  # checks if the minimum amount of variables are present
                script.update_table(l[1].upper(), l[2:])
            else:
                print('Syntax Error:', command)  # if size does not match there has to be a
    syntax error with cmd
        elif 'DELETE' in command:
            if size >= 7:  # checks if the minimum amount of variables are present
                script.delete_items(l[2].upper(), l[3:])
            else:
                print('Syntax Error:', command)  # if size does not match there has to be a
    syntax error with cmd
        elif 'SELECT' in command:
            if size >= 7:  # checks if the minimum amount of variables are present
                from_idx = l.index('from')
                script.select_specific(l[1:from_idx], l[from_idx+1].upper(), l[from_idx+2:])
            else:
                print('Syntax Error:', command)  # if size does not match there has to be a
    syntax error with cmd
        elif '.EXIT' in command:
            print('All Done')
            return
        else:  # if the command is not recognised it's and unknown command or there is
    something wring with the syntax
```

```
75          print('Syntax Error | Unknown Command')
76          print(command)
77
```

6.2 run_script.py

Contains all the functions that will execute the commands from the script.

6.2.1 create_database(self, db)

```
1          """
2          Joins the parent directory with the entered database name. Checks if the database
    already exists. If it exists
3          already it will print out and error, if not it will create a directory of the
    database name.
4          :param db: string that contains the name of the database
5          :return: None
6          """
7          path = os.path.join(self.parentDir, db)  # joins cwd and db name
8          if os.path.exists(path):  # check if path exists
9              output = '!Failed to create database ' + db + ' because it already exists'
10             print(output)
11         else:
12             os.mkdir(path)  # creates directory of path
13             output = 'Database ' + db + ' created.'
14             print(output)
15
```

6.2.2 drop_database(self, db)

```
1          """
2          Joins the parent directory with the entered database. Checks if the database already
    exists, and will either
3          error out or delete that database.
4          :param db: string that contains the name of the database
5          :return: None
6          """
7          path = os.path.join(self.parentDir, db)  # check if path exists
8          if os.path.exists(path):   # check if path exists
9              cmd = 'rm ' + '-rf ' + path  # concatenate command to input
10             os.system(cmd)  # runs the command
11             output = 'Database ' + db + ' deleted.'
12             print(output)
13         else:
14             output = '!Failed to delete ' + db + ' because it already exists.'
15             print(output)
16
```

6.2.3 use_database(self, db)

```
1      """
2      Joins the parent directory with the entered database. Checks if the database already
    exists, and will either
3      error out change the working directory to the database.
4      :param db: string that contains the name of the database
5      :return: None
6      """
7      path = os.path.join(self.parentDir, db)  # joins cwd and db name
8      if os.path.exists(path):  # check if path exists
9          os.chdir(path)  # changes cwd to this path
10         self.dbDir = path
11         output = 'Using database ' + db + '.'
```

```
12          print(output)
13      else:
14          print('Cannot Use Database | Does Not Exist')
15
```

6.2.4   create_table(self, tbl, inp)

```
1       """
2       Gets the path to that table and will check if that already exists. Then if it does not
        exist, it will change the
3       directory and call a helper function to append data to the table.
4       :param tbl:
5       :param inp: Contains all the data that will be entered into the table
6       :return: None
7       """
8       path = os.path.join(self.dbDir, tbl)  # joins cwd and db name
9       if os.path.exists(path):   # check if path exists
10          output = '!Failed to create table ' + tbl + ' because it already exists.'
11          print(output)
12      else:
13          os.mknod(path)  # creates file system of path
14          out = inp.split(',')
15          out = "|".join(out)
16          f = open(path, "a")  # opens file
17          f.write(out)  # write to file
18          f.close()  # close file
19          output = 'Table ' + tbl + ' created.'
20          print(output)
21
```

6.2.5   select_all(self, table)

```
1           """
2           Checks if the table exists and then reads all the data from the file and prints it
        out.
3           :param table: String that contains name of the table
4           :param inp:
5           :return: Check tables first * Notes
6           """
7           tbls = []
8           tbls_alias = []
9           cmd = " ".join(inp)
10          d = collections.defaultdict(list)
11
12          if 'inner join' in cmd:
13              where_idx = inp.index('on')
14              new = " ".join(inp[:where_idx])
15              new = new.replace(' inner join', ',')
16              type = 'inner join'
17          elif 'left outer' in cmd:
18              where_idx = inp.index('on')
19              new = " ".join(inp[:where_idx])
20              new = new.replace(' left outer', ',')
21              type = 'left outer'
22          else:
23              where_idx = inp.index('where')
24              new = " ".join(inp[:where_idx])
25              type = 'equal'
26          new = new.split(',')
27
28          for i, t in enumerate(new):
29              temp = t.split(' ')  # splits the table names to name and alias
30              tbls.append(temp[-2].upper())
31              tbls_alias.append(temp[-1])  # stores alias in list
32              path = os.path.join(self.dbDir, tbls[i])  # joins cwd and tbl name
```

7

```
33            if os.path.exists(path):
34                d[tbls_alias[i]] = self.read_all(path)  # stores table data into a
       dictionary key = alias, value = data
35            else:
36                print('!Failed to query table ' + table + ' because it does not exist.')
37                return
38        logic = inp[where_idx + 1:]  # contains the rest of the logic comparisons
39        res = self.join_helper(d, logic, tbls_alias, type)  # calls helper to combine tables
       , returns result
40        for line in res:
41            print(line)
42
```

6.2.6 alter_table(self, tbl, inp)

```
1         """
2         Will check if the table exists, if it doesn't exist it will print out an error.
3         If it exists it will then append the extra values to the file.
4         :param tbl: String containing name of table
5         :param inp: string that need to be inputted
6         :return: None
7         """
8         path = os.path.join(self.dbDir, tbl)  # joins cwd and db name
9         if os.path.exists(path):  # check if path exists
10            out = inp.split(',')  # takes the string a separates all the values by comma's
       and storing it into a list
11            out = "|".join(out)  # joins the list back together into a string with a '|' at
       value
12            f = open(path, "a")  # opens file
13            f.write('| ' + out)  # adds '|' to separate existing values and then writes the
       output string
14            f.close()  # close file
15            output = 'Table ' + tbl + ' modified.'
16            print(output)
17        else:
18            output = '!Failed to alter table ' + tbl + ' because it does not exist'
19            print(output)
20
```

6.2.7 drop_table(self, tbl)

```
1         """
2         Checks if the table exists and if that table exists it will delete that path. If it
       does not exist
3         it will error and print the error message
4         :param tbl: string that contains the name of the table
5         :return: None
6         """
7         path = os.path.join(self.dbDir, tbl)  # check if path exists
8         if os.path.exists(path):  # check if path exists
9             cmd = 'rm ' + '-rf ' + path  # concatenate command to run
10            os.system(cmd)  # runs the command
11            output = 'Table ' + tbl + ' deleted.'
12            print(output)
13        else:
14            output = '!Failed to delete ' + tbl + ' because it does not exists.'
15            print(output)
16
```

6.2.8 insert_table(self, table, new_data)

```
1         """
2         Takes in the name of the table and inserts the new data to that data.
```

```
3          :param table: string with name of the tale
4          :param new_data: contains the new data that needs to be inserted
5          :return: None
6          """
7          path = os.path.join(self.dbDir, table)
8          if os.path.exists(path):  # check if path exists
9              res = []
10             data = self.read_all(path)  # reads all the current data from the file and
   stores it in a list
11             for line in data:  # reads the lines and splits the data
12                 res.append(line.split('|'))
13
14             new_data = "".join(new_data)[7:-2]  # gets rid of the excess data
15             new_data = new_data.split(',')  # splits the data on commas
16             res.append(new_data)  # adds the new list to the rest of the data
17             self.insert_helper(path, res)  # calls the helper function to print the data to
   the file
18             output = '1 New Record Inserted'
19             print(output)
20         else:
21             output = '!Failed to insert into table ' + table + ' because it does not exist.'
22             print(output)
23
```

6.2.9   insert_helper(self, path, inp)

```
1          """
2          Helper that writes the new list of values to the file
3          :param path: Path that leads to the table file
4          :param inp: the new data that needs to be written to the file
5          :return: None
6          """
7          f = open(path, "a")  # opens file
8          f.truncate(0)
9          for line in inp:
10             out = "|".join(line)
11             out += '\n'  # adds the new line
12             f.write(out)  # write to file
13         f.close()  # close file
14
```

6.2.10   update_table(self, table, data)

```
1          """
2          Updates specific value if conditions match. Finds the index of the desired variable.
   Then it changes the
3          variable that needs to be changed based on the index of the variable that needs to
   change.
4          :param table: string with name of the table
5          :param data: contains the data to update the table
6          :return: None
7          """
8          path = os.path.join(self.dbDir, table)
9          if os.path.exists(path):  # check if path exists
10             var = data[1]   # gets the specific values that are in the list
11             var2 = data[5]
12             operation = data[2]
13             changeto = data[3]
14             change = data[7][:-1]
15             new, count = self.update_helper(path, var, var2, operation, change, changeto)  #
   calls helper func. to get/
16             # new data data and count of how many objects were updated
17             self.insert_helper(path, new)  # calls helper function to print new data to file
18             if count > 1:  # print plural of singular
19                 output = str(count) + ' Records Modified'
```

```
20            else:
21                output = str(count) + ' Records Modified'
22          else:
23              output = '!Failed to update table ' + table + ' because it does not exist.'
24          print(output)
25
```

6.2.11 update_helper(self, path, var, var2, operation, change, changeto)

```
1          """
2          Helper function that finds all the indexes of the values that need to be change. it
   will compare those values
3          using the operators specified and then update the specific value that matches up the
    the desired conditions.
4          :param path: string that contains the path of the file
5          :param var: contains the variable that needs to match
6          :param var2: contains the variable that needs to change
7          :param operation: the operation that needs to compare the values to satisfy the
   conditions.
8          :param change: The variable that needs to be changed
9          :param changeto: The variable that needs to be changed too
10         :return:
11         """
12         data = self.read_all(path)  # reads in all the data from the current file
13         res = [[data[0]]]  # creates the final list to be printed to file
14         curr = self.read_all_list(data)  # turns all the lines into lists making it a list
   of lists
15         count = 0  # initiates the count for the number of changes
16         where_idx = self.find_idx(curr[0], var2)  # finds the index that needs to found
17         set_idx = self.find_idx(curr[0], var)  # finds the index that equates to being
   changed
18         if operation == '=':
19             operation = '=='
20
21         for line in curr[1:]:
22             if operation == '==':
23                 if eval('line[where_idx]' + operation + 'change'):  # if the variable
   satisfies the condition to change
24                     line[set_idx] = changeto  # changed the desired variable that needs to
   be changed
25                     res.append(line)
26                     count += 1
27                 else:
28                     res.append(line)
29             elif operation == '!=':  # checks if operation is not equal because of
   differences in float and strings
30                 if eval('line[where_idx]' + operation + 'change'):  # if the variable
   satisfies the condition to change
31                     res.append(line)
32                 else:
33                     line[set_idx] = changeto  # changed the desired variable that needs to
   be changed
34                     res.append(line)
35                     count += 1
36             else:
37                 if eval('float(line[where_idx])' + operation + 'float(change)'):  # <, > can
    only compare numbers
38                     line[set_idx] = changeto  # changed the desired variable that needs to
   be changed
39                     res.append(line)
40                     count += 1
41                 else:
42                     res.append(line)
43
44         return res, count
45
```

6.2.12 delete_items(self, table, data)

```python
"""
Takes in the table and deletes the desired rows that satisfy the conditions.
:param table: string with name of table
:param data: contains the data and conditions needed to delete a row
:return: None
"""
path = os.path.join(self.dbDir, table)
if os.path.exists(path):  # check if path exists
    where = data[1]
    if data[2] == '=':  # turns it into and '==' to use as an operator
        operation = data[2] + data[2]
    else:
        operation = data[2]
    var = data[3][:-1]
    new, count = self.delete_helper(path, where, operation, var)  # calls helper to get new data and count
    self.insert_helper(path, new)  # calls helper to print new data to file
    if count >= 1:  # determines if singular or plural print
        output = str(count) + " Records Deleted"
    else:
        output = str(count) + " Record Deleted"
else:
    output = '!Failed to delete items in table ' + table + ' because it does not exist.'
print(output)
```

6.2.13 delete_helper(self, path, where, operation, var)

```python
"""
Helper to delete where it find the index of the variable that needs to compared. Then it will compare it with
the operation specified to determine if that row needs to be deleted.
:param path: string that has the path of the table
:param where: the variable that needs to satisfy the condition
:param operation: the logic operation
:param var: the variable that needs to match the where to delete the row
:return: the new list that needs to be printed and the count of the items deleted.
"""
data = self.read_all(path)  # reads all the data that is stored in the current file
res = [[data[0]]]   # initiates the new data with variable names
curr = self.read_all_list(data)  # turns all the lines into a list
where_idx = self.find_idx(curr[0], where)  # finds the index where the variable that needs to check is
count = 0
for line in curr[1:]:
    if operation == '==':  # checks if operation is equal because of differences in float and strings
        if eval('line[where_idx]' + operation + 'var'):  # eval turns string into a logic comparison
            count += 1
            continue
        else:
            res.append(line)
    elif operation == '!=':  # checks if operation is not equal because of differences in float and strings
        if eval('line[where_idx]' + operation + 'var'):
            res.append(line)
        else:
            count += 1
    else:  # checks if operation is equal because of differences in float and strings
        if eval('float(line[where_idx])' + operation + 'float(var)'):  # greater or less than operations for num
```

```
29                    count += 1
30                    continue
31              else:
32                    res.append(line)
33        return res, count
34
```

6.2.14 select_specific(self, var, table, where)

```
1         """
2         Selects a specific row depending on the where condition and the variables it wants
     to see.
3         First deletes the the values that depend on the where clause. Then goes through and
     deletes the columns
4         that are not needed to be seen.
5         :param var: contains the variable for the where clause
6         :param table: contains the name of the table in string
7         :param where: contains the string of the conditions that needs to be satisfied
8         :return:
9         """
10        obj = where[1]
11        op = where[2]
12        w = where[3][:-1]
13        path = os.path.join(self.dbDir, table)
14        if os.path.exists(path):   # check if path exists
15            data = self.read_all(path)
16            missing = []
17            variables = self.get_curr_var(data[0])
18            for i in range(len(var)):
19                var[i] = var[i].replace(',', '')   # gets rid of all commas
20            for v in variables:
21                if v not in var:   # checks if the variable needs to be seen
22                    missing.append(v)
23            res = self.select_specific_helper(path, missing, data, obj, op, w)   # calls
     helper to delete rows and cols
24            for line in res:
25                print("|".join(line))
26        else:
27            output = '!Failed to select items in table ' + table + ' because it does not
     exist.'
28            print(output)
29
```

6.2.15 select_specific_helper(self, path, missing, data, obj, op, w)

```
1         """
2         It will call the delete helper to get rid of rows that don't have the specific value
     .
3         Finds the missing values that are not needed to be displayed. It will delete those
     columns that are not
4         needed.
5         :param path: String with Path to the table
6         :param missing: Contains the list of values you want
7         :param data: list that has the data from the file
8         :param obj: the object that wants to be found
9         :param op: the logic operator
10        :param w: the values that needs to be compared with the operator
11        :return: return the list that needs to be inputted into the file
12        """
13        res = []
14        data, count = self.delete_helper(path, obj, op, w)   # delete specific rows that
     satisfy condition
15        for line in data:
16            new = "| ".join(line)
17            res.append(new.split('| '))
```

```
18          for miss in missing:
19              where_idx = self.find_idx(res[0], miss)
20              for line in res:
21                  del line[where_idx]  # deletes the column from all the lines
22          return res
23
```

6.2.16   read_all(self, path)

```
1           """
2           Checks if the table exists and then reads all the data from the file and prints it
      out.
3           :param path: String that contains path of the table
4           :return: None
5           """
6           self.data = []
7           if os.path.exists(path):  # check if path exists
8               with open(path) as file_in:  # starts reading from file
9                   for line in file_in:
10                      self.data.append(line.rstrip())
11              return self.data
12          else:
13              output = '!Failed to query table because it does not exist.'
14              print(output)
15
16
```

6.2.17   read_all_list(self, data)

```
1           """
2           Turns a string into a list
3           :param data: Data with strings in list
4           :return: return a list with the strings separated into lists
5           """
6           curr = []
7           for line in data:
8               curr.append(line.split('|'))
9           return curr
10
```

6.2.18   get_curr_var(self, data)

```
1           """
2           Splits the input data and removes commas in list leaving only the variable names
3           :param data: List of data from the file
4           :return: A new list that contains only the variable names
5           """
6           res = []
7           new = data.split('| ')  # splits into list by '|'
8           for var in new:
9               s = var.split(' ')  # splits into a list generated by spaces
10              s[0] = s[0].replace(',', '')  # removes commas
11              res.append(s[0])
12          return res
13
```

6.2.19   find_idx(self, inp, var)

```
1           for i in range(len(inp)):
2               if var in inp[i]:
3                   return i
4           return -1
5
```

6.2.20   inner_join(rows, res, d, tbls_alias, cols_id)

```
1            """
2            Will perform the inner join by concatinating all the values that match the logic
     operation
3            """
4            for i in range(1, rows):  # compare all the rows and if the variables being compared
      match join them together
5                for j in range(1, rows):
6                    d1 = d[tbls_alias[0]][i]
7                    d2 = d[tbls_alias[1]][j]
8                    if d1.split('|')[cols_id[0]] == d2.split('|')[cols_id[1]]:  # compare
     variables
9                        res.append(d1 + '|' + d2)
10           return res
11
```

6.2.21   outer_join(rows, res, d, tbls_alias, cols_id)

```
1            """
2            Will perform the inner join by concatenating all the values that match the logic
     operation
3            """
4            for i in range(1, rows):  # compare all the rows and if the variables being compared
      match join them together
5                found = False  # determines if they found a match
6                for j in range(1, rows):
7                    d1 = d[tbls_alias[0]][i]  # gets the data in the first table
8                    d2 = d[tbls_alias[1]][j]  # gets the data in the second table
9                    if d1.split('|')[cols_id[0]] == d2.split('|')[cols_id[1]]:  # compare
     variables
10                       res.append(d1 + '|' + d2)
11                       found = True
12               if not found:
13                   res.append(d[tbls_alias[0]][i] + '|' * len(cols_id))
14           return res
15
```