# CS457 Project 1

Yifeng Qin

9/24/20

## 1 How does my program organize multiple databases

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

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 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.

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.

# 4 Functions

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

## 4.1 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.

### 4.1.1 read_file()

```
1
2       def read_file():
3           """
4           Opens the file with standard input, and reads every line and checks if it is a valid
     command to be
5           appended to the list
6           :return: Returns a List of the Commands
7           """
8           commands = []
9           for line in sys.stdin:
10              if line == '\r\n' or line[0:2] == '--':  # checks if it is a empty new line or
    it starts with '--'
11                  continue
12              else:
13                  commands.append(line.rstrip())  # removes newline and special characters
    from line to append to list
14
15          return commands
16
```

### 4.1.2 run_commands()

```
1       def run_commands():
2       """
3       Calls the helper function read_file to get all the valid commands from the file. It will
         run through the list and
4       splice the commands to get rid of special characters.
5       :return: None
6       """
7       commands = read_file()  # calls helper function to initialize to list commands
8       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])  # 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])   # 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])   # 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 'USE' in command:
            if size == 2:  # checks if all arguments are present
                script.use_database(l[1][:-1])   # 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], 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]) # 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], 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 '.EXIT' in command:
            return
        else:  # if the command is not recognised it's and unknown command or there is something wring with the syntax
            print('Syntax Error | Unknown Command')
            print(command)

```

## 4.2 run_script.py

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

### 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.
        :param db: string that contains the name of the database
        :return: None
        """
        path = os.path.join(self.parentDir, db)  # joins cwd and db name
        if os.path.exists(path):  # check if path exists
            output = '!Failed to create database ' + db + ' because it already exists'
            print(output)
        else:
            os.mkdir(path)  # creates directory of path
            output = 'Database ' + db + ' created.'
            print(output)

```

### 4.2.2 drop_database(self, db)

```
        """
        Joins the parent directory with the entered database. Checks if the database already
     exists, and will either
        error out or delete that database.
        :param db: string that contains the name of the database
        :return: None
        """
        path = os.path.join(self.parentDir, db)  # check if path exists
        if os.path.exists(path):   # check if path exists
            cmd = 'rm ' + '-rf ' + path  # concatenate command to input
            os.system(cmd)  # runs the command
            output = 'Database ' + db + ' deleted.'
            print(output)
        else:
            output = '!Failed to delete ' + db + ' because it already exists.'
            print(output)

```

### 4.2.3 use_database(self, db)

```
    """
    Joins the parent directory with the entered database. Checks if the database already
    exists, and will either
    error out change the working directory to the database.
    :param db: string that contains the name of the database
    :return: None
    """
    path = os.path.join(self.parentDir, db)  # joins cwd and db name
    if os.path.exists(path):  # check if path exists
        os.chdir(path)  # changes cwd to this path
        self.dbDir = path
        output = 'Using database ' + db + '.'
        print(output)
    else:
        print('Cannot Use Database | Does Not Exist')

```

### 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 call a helper function to append data to the table.
    :param tbl:
    :param inp: Contains all the data that will be entered into the table
    :return: None
    """
    path = os.path.join(self.dbDir, tbl)  # joins cwd and db name
    if os.path.exists(path):   # check if path exists
        output = '!Failed to create table ' + tbl + ' because it already exists.'
        print(output)
    else:
        os.mknod(path)  # creates file system of path
        out = inp.split(',')
        out = "|".join(out)
        f = open(path, "a")  # opens file
        f.write(out)  # write to file
        f.close()  # close file
        output = 'Table ' + tbl + ' created.'
        print(output)

```

#### 4.2.5 select_all(self, table)

```
        """
        Checks if the table exists and then reads all the data from the file and prints it out.
        :param table: String that contains name of the table
        :return:
        """
        path = os.path.join(self.dbDir, table)  # joins cwd and db name
        if os.path.exists(path):  # check if path exists
            with open(path) as file_in:  # starts reading from file
                for line in file_in:
                    self.data.append(line.rstrip())
            for line in self.data:  # prints data to terminal
                print(line)
            self.data = []
        else:
            output = '!Failed to query table ' + table + ' because it does not exist.'
            print(output)

```

#### 4.2.6 alter_table(self, tbl, inp)

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

```

#### 4.2.7 drop_table(self, tbl)

```
        """
        Checks if the table exists and if that table exists it will delete that path. If it
    does not exist
        it will error and print the error message
        :param tbl: string that contains the name of the table
        :return: None
        """
        path = os.path.join(self.dbDir, tbl)  # check if path exists
        if os.path.exists(path):  # check if path exists
            cmd = 'rm ' + '-rf ' + path  # concatenate command to run
            os.system(cmd)  # runs the command
            output = 'Table ' + tbl + ' deleted.'
            print(output)
        else:
            output = '!Failed to delete ' + tbl + ' because it does not exists.'
            print(output)

```