**CSCE 4523 Database Management Systems**

**Homework 1**

**By: Bryan Williams, Tony Sanchez**

**Objectives**

The objectives of the assignment between Homework 1 Part 1 and Homework 1 Part 2 were to get familiar with working with database management system techniques to implement a database management system. Part one was more introductory than part two; we were given starter code to see what our program should look like, what methods to create, what parameters they take in, and what they return. At the end of part one we should have been able to take in some form of data from a csv file and return it in a created data file as well as a config file that displays how many records and the record size of the data file. After getting part one up and running, the objective of part two was to clean up our code but also continue working with the program to implement a program that finds records through id number, display, update, and delete them. Furthermore, we were expected to rigorously test our code to ensure that we address any bugs in our code.

**Approach**

We started off using the sample code for python that was provided to us. It was more of a foundational support for our code rather than a copy and paste of all the methods. Our design strategy was influenced by the homework’s instructions as it laid out a clear structure to how and what methods we should implement. We started off by creating the menu that would be the interface that the user interacts with. From there, we created each of the methods in the order that it was presented in the menu. This strategy seemed to work well since the methods used to carry out the lower-numbered menu options were also needed to carry out the higher-numbered menu options.

A screenshot of a computer

Description automatically generated The first menu option allows the user to create a database, which in return created a data file (.data) and a configuration file (.config). This process is marked as the foundation of the Database Management System. It prompts the user to enter the name of the comma-separated-values file (.csv). Once they enter a valid file name, the program generates a config file and a data file. The config file keeps up with the number of records and the total record size. We decided to set our record size to 81 because it gave each of the columns in the database enough space to fit all the content, but it also doesn’t render the data to be too spaced apart. In terms of how much space each of the columns got, we decided to give the id numbers 10 spaces, the state and city names 15 spaces, and lastly the name of the schools got 40 spaces. A small sample from our data file is presented below in Figure 1.

Figure 1: A sample of the formatted database from our data file

A screen shot of a computer

Description automatically generatedIn addition to creating the data file, the create option also creates the config file, which I stated previously, that contains the number of records and the record size. This information will be needed for opening the database in the future. Our config file is presented below in Figure 2.

Figure 2: Image of our generated config file

**Results**

As we created the different methods each method had different error handling requirements. The create-new-database option had to make sure we had an existing file that could be converted into a database. The open-database option had to make sure that there was an existing data file to open, and if there wasn’t, the user should be prompted that there is no existing database. The editing methods had to make sure that there were records to be able to edit and return an error message to the user if none were found. In order to make this possible, we decided to return Booleans or integers. If the method returns true or 1 then it signifies that everything worked, but if it returned false or -1 then it meant that something went wrong. For the most part, the error message usually states that there is “id # is not found” or “record # is not found.” Overall, the efficiency of the program is good. The time complexity for searching for a specific record is log(n) because we used binary search to search the data file for the inputted id. Throughout the entire program, the id number is used as the primary key to make it easier to find specific records.

One method that we attempted, that didn’t end up working out, was storing a list of deleted records to make it easier when printing out the report, which was written to print out the first 10 valid records. This method didn’t work because we would have needed to always keep track with this list, but the only issue was, if the user were to end the program, the list would be picked up by the trash cleaner in memory. The only option would be to store this list in the config file, but that wasn’t efficient. What we decided to do was to skip over the records that had an empty string in place of the state column when printing out the report.

Another issue we ran into was updating the config file. When adding a record, all we needed to do was increase the number of records in the config file which was simple, but we couldn’t do the same for removing a record. This is because when we remove the record, we keep the id but delete the content from the other attributes in the record. This meant that although we were removing the information, we weren’t actually removing that entire record from the file, which means the number of records remained the same even after deleting one.

Since we worked in pairs, I was in charge of the methods that were created in the Database class and Tony was in charge of creating the methods in the other file. We both were in charge of testing our own methods, but also testing the functionality of the program as a whole.

**Testing**

We tested our program by entering the general, valid inputs, but also entering invalid inputs. We entered the valid inputs for ensure the program worked the way that it was created too. We used the invalid inputs to ensure that there was error handling, and to sure that the program didn’t work in ways that it shouldn’t have. For example, we entered names of files that didn’t exist, and we entered id numbers that weren’t included in the data file. Our main method of checking our program was the use of menu option 4 which was the “display record” option. This option was an easy way to check that things were working the way they should. We tested if a file was correctly removed from the data base with the display option. We also used the display record to ensure that when we updated a record, the changes were reflected in the data file. Another way we checked the removal of a record was the “Print Report” option in the menu because if a record was successfully removed, then it should be skipped and therefor shouldn’t show up in the report. Additionally, for the update record option, we decided to not give the user the option to change the id of the record since that is the primary key and shouldn’t be changed.