Java Assignment 2 – Homemade Database System

This report is intended to detail the stages of development for the second Java assignment. Each stage of development has been put into a separate folder and further details are included in a mini-report .txt file for each phase.

Stage 1 – Creating Record

The first stage of development involved creating records. It was unclear at this stage what a record should be, except that it is a mechanism that will hold Strings. The Record.java file in this stage, therefore, had two ways of storing records, as an array of Strings or as an ArrayList of Strings.

The methods at this stage simply obtained the record, displayed the record, found the length of the record, and set the Record.

Stage 2 – Creating Tables

At the next stage of development, when tables became involved, it was clearer what the objective of Records was, and the array of Strings methods were removed. Now, records were simply ArrayList of Strings, and can be added to tables, which are ArrayLists of ArrayLists of Strings. In retrospect, the Table class could have held an arraylist of Records (List<Record> table) rather than the 2D arraylist (List<List<String>>). This would have made the program a lot clearer to the reader and would have demonstrated the purpose of the Record class more clearly.

The methods in the Table.java class allowed creation of tables, deletion of records, selection of records, and also allowed a field in a record to be updated. At this point, records were referenced by row and column; it wasn't until a later stage that these records were referenced by Keys (Keys.java).

A lot of the testing at this stage remained minimal as the direction the program was going remained unclear. Moreover, most of the methods created in this version of the program returned void (it wouldn't be till later that these methods returned values – and therefore made them easier to Unit test). Considerably, integration tests between these methods were conducted at this stage.

Stage 3 and 4 – File manipulation

These stages involved reading and writing to files. The first stage (stage 3) involved reading from the file. This phase took a long time as a lot of changes were made before an agreed upon method was created. The program only read readable text files, making all other files invalid. A text file could only be readable if there was a ':' character in it. This is because ':' characters were used at this stage to distinguish between fields in the records. A field is separated by a ':' character to allow for commas, spaces, etc. to exist in fields. NOTE: it wouldn't be till a later stage that this would be edited to allow for ':' characters to be in fields – done so by changing the control character(s) to '/:'.

Issues that arose at this stage of development was the newline character when writing to a file. This had to be added so that the program knew when to stop reading in a record when the newly created file was read back into the program.

Stage 5 – Adding keys, print functions, and major refactoring

This was the largest stage of development. In this phase, two extra classes, Key.java and Print.java, were added and allowed the user to set columns to include keys and for tables to be displayed to the terminal, respectively.

The Print class allowed the user to print a table to the terminal, with its columns lined up and wrapped in dash ('-') and bar ('|') characters – to make them more presentable. The tables can also change in size depending on the largest field in that particular column.

The Key class merely searched for duplicates in the column which is set as the key column. The other major changes occurred in Table.java where most of the methods were changed to return values (making them easier to test) and also to account for these key columns.

Most of the unit testing was done at this stage as major refactoring took place.

Stage 6 – Final stage of development

In the final stages of development, the control character was changed from ':' to '/:' to allow for colon characters in fields. This is a design choice that should have been made earlier as this involved going back and changing all instances of the ':' character, which proved to be tedious and even troublesome as unit tests and test files needed to be changed also.

In addition, this stage involved creating the Database class (for the purpose of being a database rather than as a control class). This class wrapped up the ideas developed in the previous classes and allowed tables to be added, removed, updated, and selected – thereby allowing multiple tables to coexist at the same time using the single Database object.

Extensions

User Interface

For the extension task, a User interface was implemented in the UI.java file. This simply allowed the user to manipulate the database in a similar way to the Database.java file – it took this logic and applied it to include user input.

The user is first asked to give the name of their database and then can run a series of commands from there. All the commands are not case sensitive, so “CREATE” has the same meaning as “create”. They can type “options” / “OPTIONS” to find out the keywords and how the commands should be structured.

The commands are “CREATE”; “INSERT”; “SELECT”; “PRINT”; “SHOW”; “UPDATE”; “DELETE”; “UPLOAD”; “WRITE”; and “EXIT” – the latter being used to terminate the program.

The “CREATE” statement allows the user to create a table. The structure of the statement is “CREATE tableName field1 /: field2 /:” – this will create a table called “tableName” and add two fields called “field1” an “field2” as the column names. Noticeably, the user has to enter ‘/:’ in order to separate columns. This is so they can enter multiple spaces, commas, colons, etc.

They can also set the key column in the create statement by using the “/SET” keyword. For example, “CREATE tableName field1 /: field2 /SET 1” will make field2 the key column. From then on, when a value is inserted into the table there cannot be duplicates in this column

The “UPLOAD” statement allows the user to upload tables from readable text files. The structure of the statement is “UPLOAD tableName filename.txt”, which will create a table called “tableName” from the file “filename.txt” which will have to have fields separated by ‘/:’ characters. Like with the create statement, the key column can be set for this command also: “UPLOAD tableName fieldname.txt SET 1 will make the 2nd column in the table the key column.

The “INSERT” statement allows the user to insert values into an (existing) table. The structure of the statement is “INSERT tableName value1 /: value2”. This will add value 1 to the first column and value 2 to the second column. The statement is only valid is the table name exists, and the columns are the same length as the header (e.g. if there were two columns in the table and the user tried to enter three values then an error would be thrown).

The ”SELECT” statement selects records based on a key value or a row in an existing table. The structure of the statement is “SELECT tableName row 1”. This will select the first record in the table “tableName”. They can also select based on key values, e.g. say there was a value “re16621” in the table “tableName”, then typing “SELECT tableName re16621” should suffice to return the entire record where “re16621” is the value in the key column. NOTE: If the record isn’t found “empty set” will be returned by the function and then an error will be thrown.

The “DELETE” statement deletes a specific record based on its key value or a row in an existing table. The structure of the statement is “DELETE tableName row 1”. This will delete the first record in the table. Furthermore, “DELETE tableName re16621” will delete the row where re16621 is the value in the key column.

The “UPDATE” statement updates a field in a record in an existing table. E.G. “UPDATE tableName row 1 col 1 newvalue” will update the value in the first row and second column with “newvalue”. Moreover, the user can select based on the value in the key column and then the value within that record, e.g. say there is a record where the value in the key column is re16621 and the value in the second column is “Rich” then the user can enter “DELETE re16621 /: Rich /: newValue”. This will remove the value Rich and replace it with “newValue”.

The “SHOW” statement shows the tables within the database. It simply returns the table names, key column number and number of records and prints them to the screen within a stylised table. The statement only takes one value, “SHOW”, and nothing else.

Penultimately, the “WRITE” statement allows the user to write an existing table to a readable text file (so that the user can store the tables they have created and edited and use them another time).

Finally, the “PRINT” statement prints the table to the terminal. The statement structure is “PRINT tablename” and nothing else. This will print the table “tableName” (providing it exists). Moreover, the key column will be highlighted by an ‘*’ (asterisk) character.