Software Developer Technical Test

Andrei Popa
ap4u19@soton.ac.uk

February,24,2020

Contents

1. Abstract

For the implementation of my solution I have used the following technologies: XAMPP: Apache, MySQL, phpmyadmin. I have created a database design with 5 tables, having a one-by-many relationship between them. One table is an intermediary table, resulted from breaking down a many-by-many relationship. For inserting the data into the database, I have used a stack ADT and created a tree where all nodes can either be a parent node(dir) or a child node(dir/subdir). For the last question I have implemented the Breath-first search algorithm with a custom adaptation for printing out the path to the searched element.

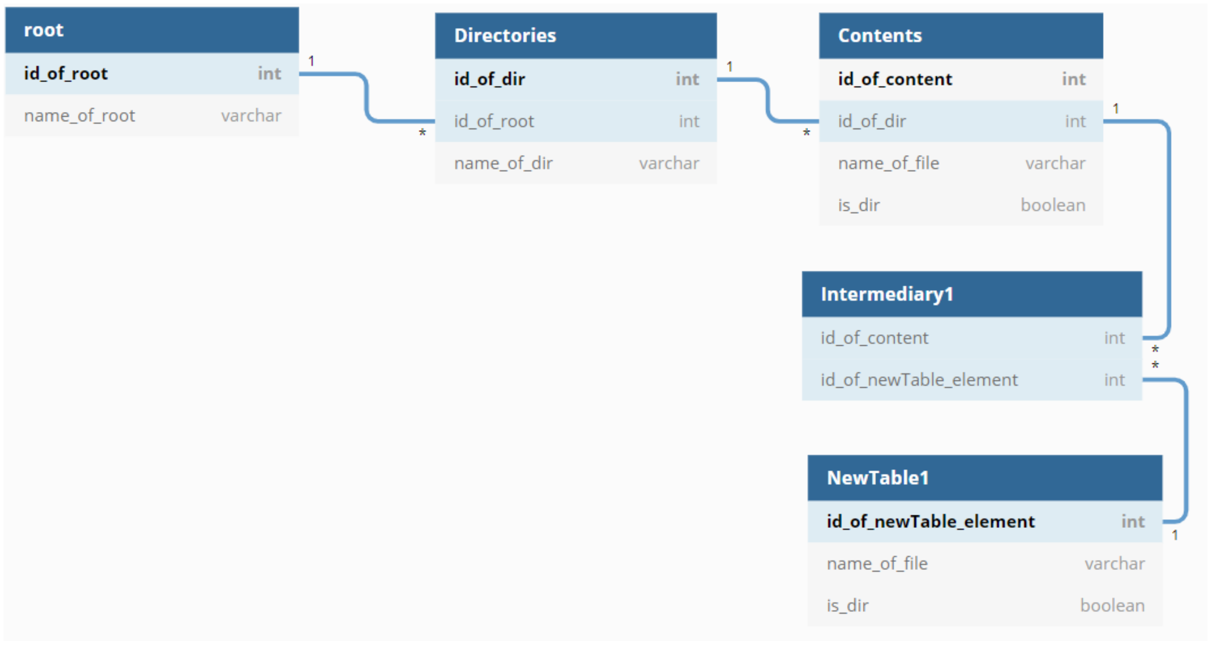
2.Introduction

The aim of this report is to describe my implementation for the Software Developer Technical Test that consists of

1. Creating a database design able to store a given file system structure.
2. Creating a text file with the given file system structure and insert into the Database using PHP programming language.
3. Providing other possible solutions for the above given points.
4. Creating a web interface that allows the user to search into the database using an input box and a search button.

3.Main Body

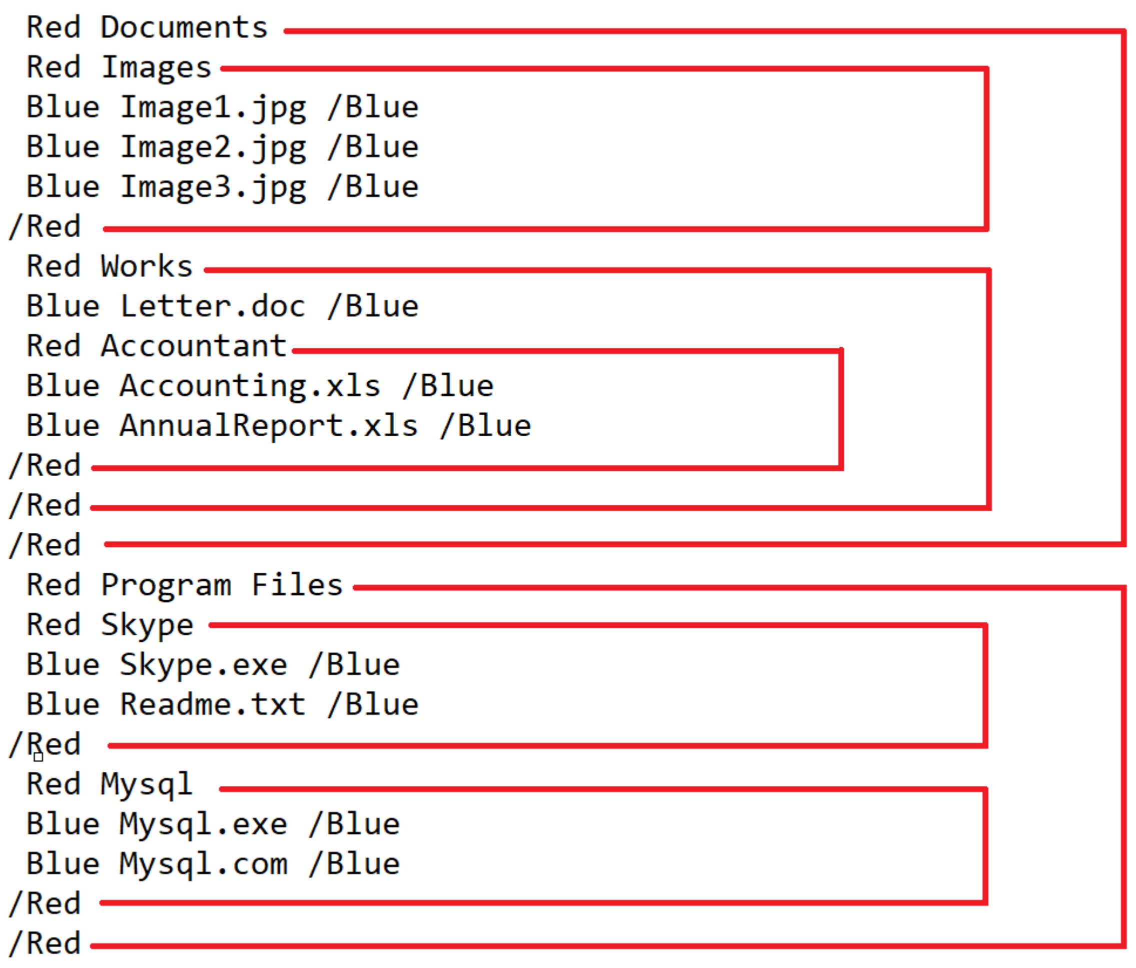
3.1 Database Design implementation



The given file structure is a Tree structure and, because of this, the root table will contain Documents and Program Files. The directories are the next level of the tree, consisting of the sub-directories of the root directory and these nodes are put in the Directories table. Between the root and Directories table is a one-by-many relationship, since every root can have many other sub-directories, but a sub-directory can have only one root. The third level of the tree consists of files and directories and, because of this, I added a new filed is_dir that is True when the added node is a directory and False when it is a file. Between the Directories and Contents table is a one-by-many relationship. If the added element in Contents has the is_dir set to True, I create a new table(NewTable1) for the elements of that table. In this case we have a many-by-many relationship, because for every dir element in the Contents table we would have many sub-files and sub-directories. Hence, the many-by-many relationship will be broken down to 2 one-by-many relationships using an intermediary table (Intermediary1) whose PrimaryKey consists of both of its fields.

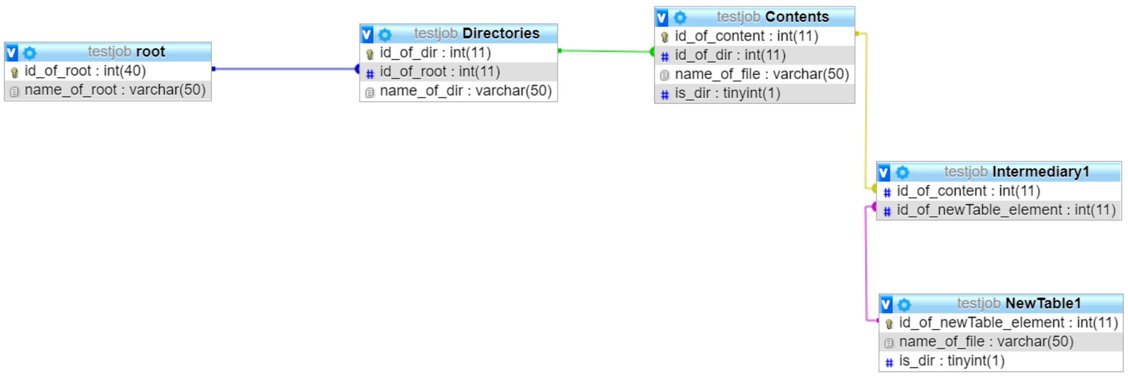
3.2 Input from a text file

I created a .txt file named InputFile with the following structure:

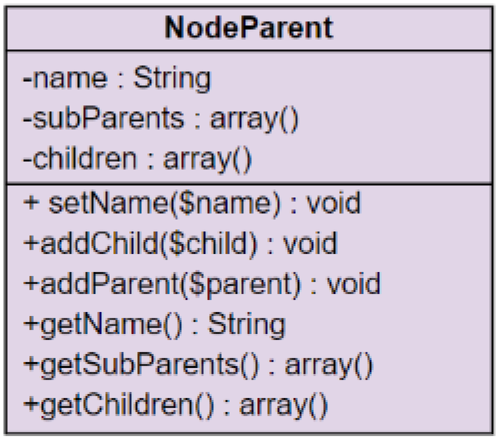


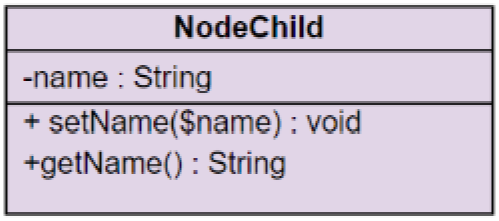
```
 Red Documents
 Red Images
 Blue Image1.jpg /Blue
 Blue Image2.jpg /Blue
 Blue Image3.jpg /Blue
/Red
 Red Works
 Blue Letter.doc /Blue
 Red Accountant
 Blue Accounting.xls /Blue
 Blue AnnualReport.xls /Blue
/Red
/Red
/Red
 Red Program Files
 Red Skype
 Blue Skype.exe /Blue
 Blue Readme.txt /Blue
/Red
 Red Mysql
 Blue Mysql.exe /Blue
 Blue Mysql.com /Blue
/Red
/Red
```

The created database system in phpmyadmin:



My implementation is to create 2 Tree structures with each root being a dir (Documents and Program Files), where each node can be either a file or a directory. Since every node can either be a parent node or a child node, I designed a NodeParent class and a NodeChild class for inserting the data from inputFile.txt following the diagrams:





My algorithm for inserting the data into the database is using the above classes and a stack ADT:

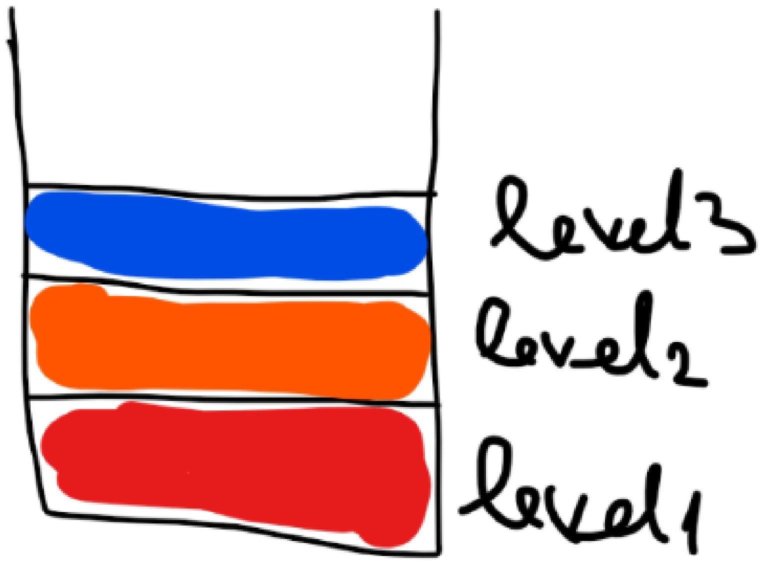
I am reading the .txt file line by line and if I find “Red” in the beginning of the line I create a new NodeParent object and push() it in a Stack, having two cases:

1. The stack is empty. This means that the NodeParent object we are adding is the root of the tree.
2. The stack is not empty. This means that the NodeParent object we are adding is the root of a sub-tree. Hence, I peek() to get the NodeParent element from the stack, I add the new NodeParent object to the subParents array of the root and we push the element into the stack.

If I find ”/Red” in the beginning of the line, I pop() the stack.

If I find “Blue” in the beginning of the line, I create a new NodeChild object, I peek() to get the top of the stack, and add that child to the returned NodeParent object

Each level of the stack that was created will have a NodeParent object that represents a directory.



Adding the NodeParent nodes into the database:

1. The element that is in the first level of the stack will be inserted into the root table.
2. The element that is in the second level of the stack will be inserted into the Directories table.
3. The element that is in the third level of the stack will be inserted into Contents table with the Boolean flag is_dir set to true.

Adding the NodeChild nodes into the database:

1. When the stack is on the second level and we create a new ChildNode object, I insert the child into the Contents table with the Boolean flag is_dir set on false.
2. When the stack is on the third level and we create a new ChildNode object, I insert the child into the NewTable1 with the Boolean flag is_dir set on false.

For the one-by-many relationships I use _SESSION[“<value>”] to store the last inserted id into the database. For determining the last inserted id I use $mysqli->insert_id.

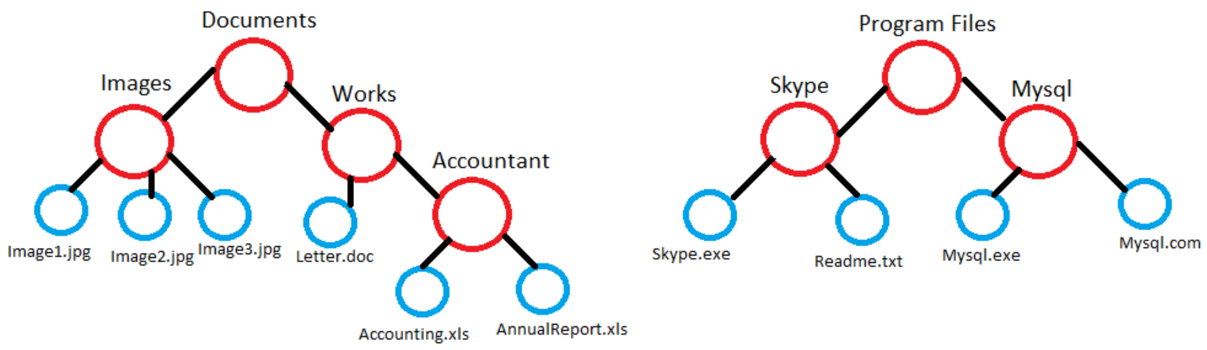
3.3 Other possible solutions.

Another possible solution to store the last ID inserted into the DB would have been using the SCOPE_IDENTITY() for the SQL server or LAST_INSERTED_ID(), but calling it requires extra query, which is slower. Also, The MySQLi approach is a more object-oriented approach.

Another possible solution to create the .txt structure would have been YAML, but for this small sample file structure it does not matter which method I choose.

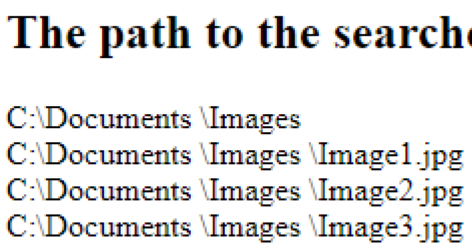
3.4 **Web interface and algorithm for the searching tool**

Due to my implementation, now I have the following Tree structures:



I use the Breadth-first search algorithm to traverse each tree, with a custom adaptation. If I find a possible match for the word there are 2 cases:

1. If that found node is a NodeChild object, I print the path from the root of the tree to that node.
2. If that node is a ParentNode object, I print the path from the root to that node. If the node has other nodes linked to it, I print he path from the root for each one of tem.



The path to the searched item image :

```
C:\Documents \Images
C:\Documents \Images \Image1.jpg
C:\Documents \Images \Image2.jpg
C:\Documents \Images \Image3.jpg
```

Exception Handling for Q4

The application does not allow the user to submit without entering any character.

The application will output “Invalid input: only spaces!” if the user submits only space characters

The application will output “The searched item <name_of_item>” was not found if the user searches for an element that does not exist.

The application will eliminate unnecessary spaces (i.e. “   image”) to search through the Tree.