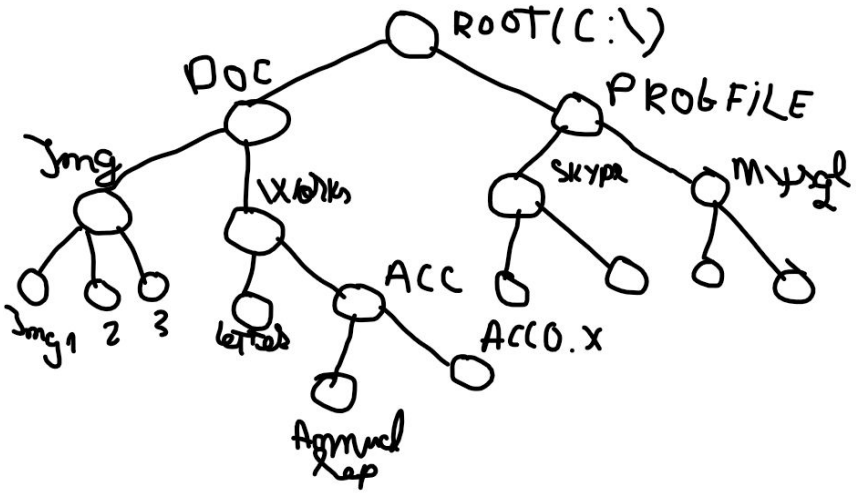
Report about the exercise sheet

C:\

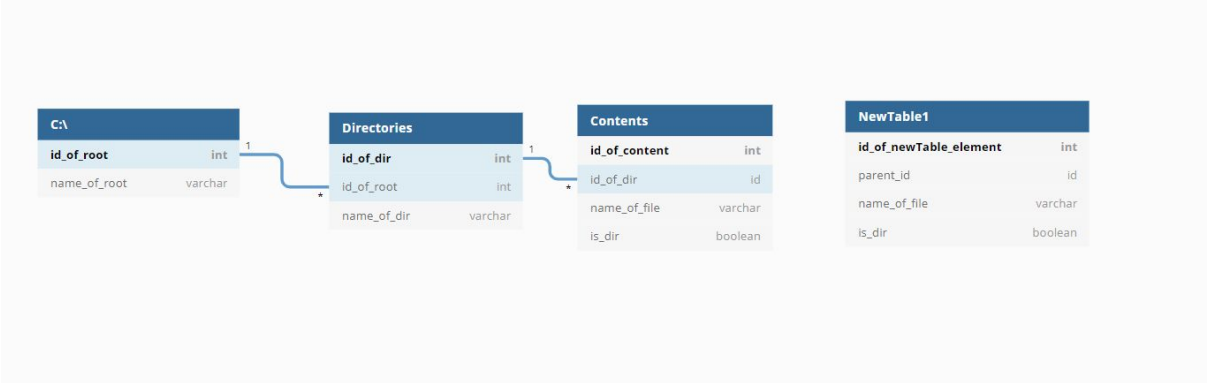
- Documents
  - Images
    - Image1.jpg
    - Image2.jpg
    - Image3.png
  - Works
    - Letter.doc
    - Accountant
      - Accounting.xls
      - AnnualReport.xls
- Program Files
  - Skype
    - Skype.exe
    - Readme.txt
  - Mysql
    - Mysql.exe
    - Mysql.com

I can see it like a tree:

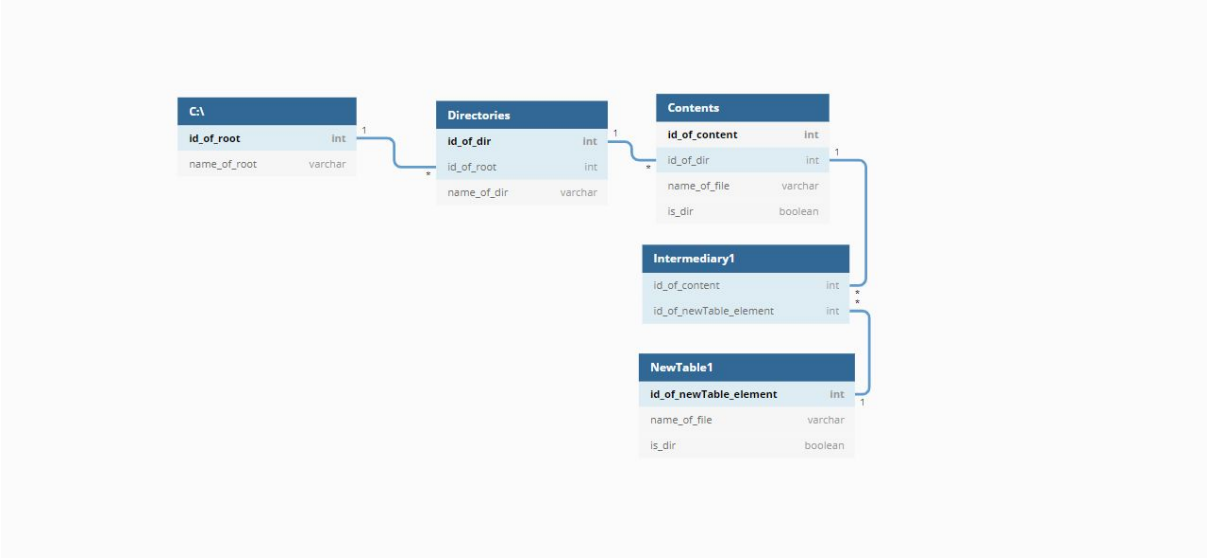


Report about the exercise sheet

Q1.Database Design implementation:

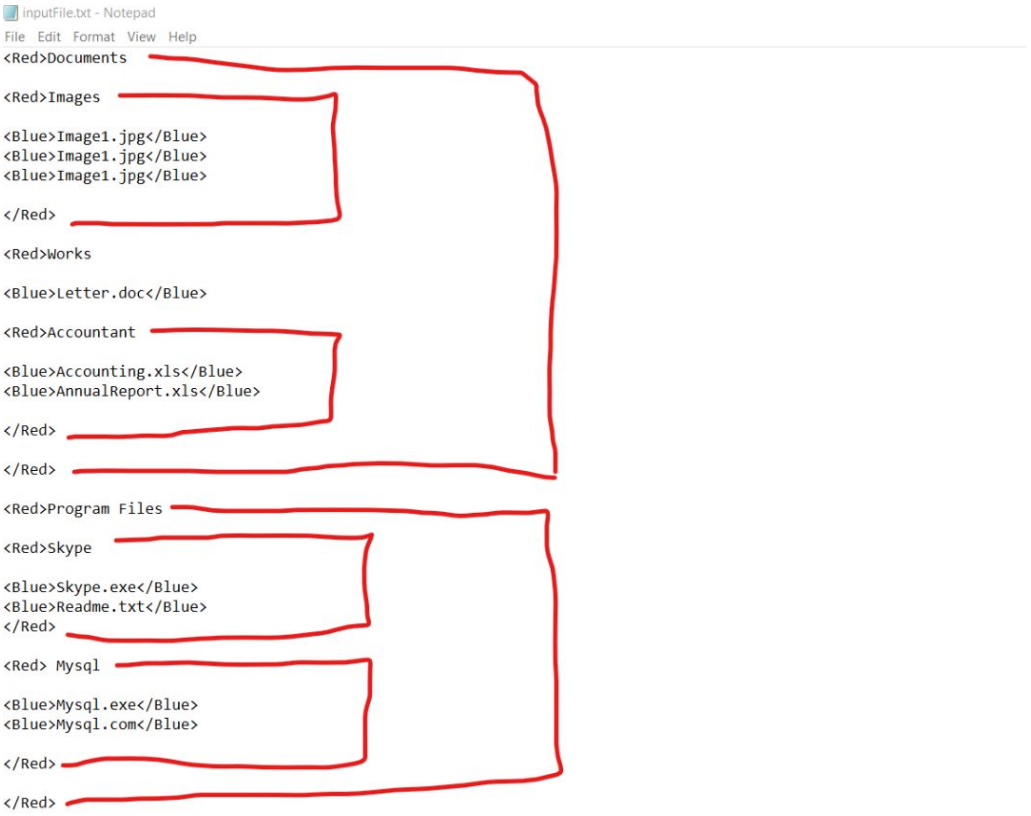


My idea is that whenever we find an element from components to be a dir, we will set the is_dir boolean field to True, and then we will create a new table(NewTable1) for that table that contains the dir element with the relation one-to-many. But, since every element from contents can be a dir and for each dir we would have many children available, there is a many-by-many relation, that will be broken down to 2 one-by-many relationships using an intermediary database:



Q2.Input from text file

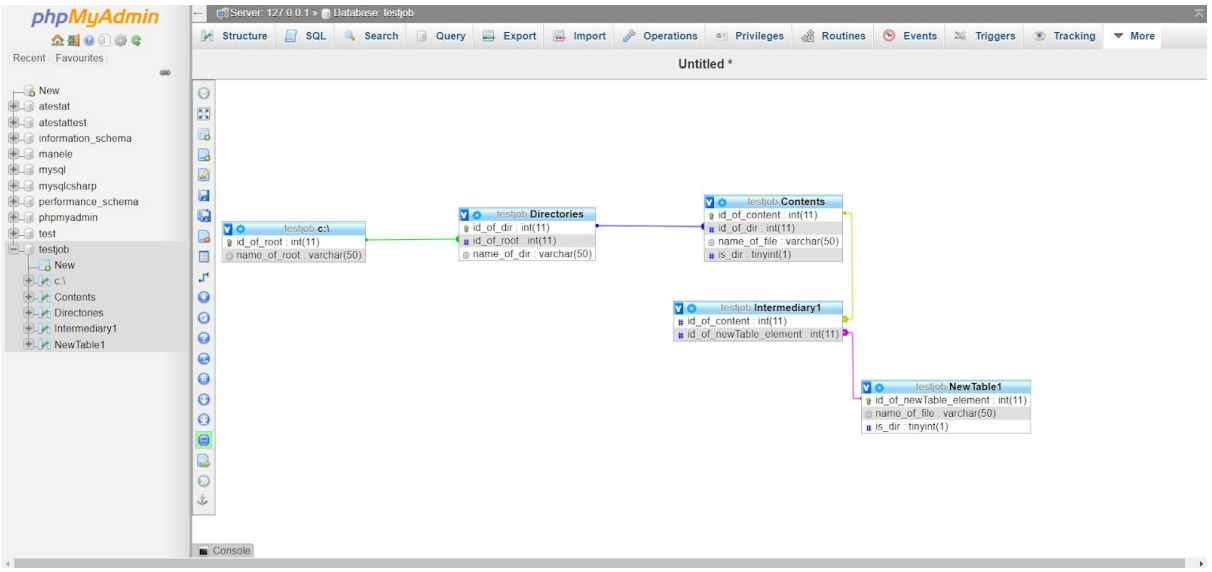
My first idea to insert the data into the database I thought of using a method involving the Markup Language for the inputFile.txt:



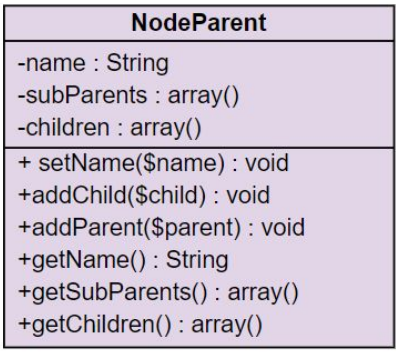
But later I found out that php ignores the <> tags because it considers them as html tags.Hence, I changed the inputFile.txt

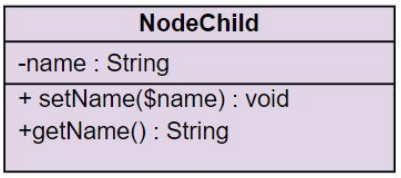
Report about the exercise sheet

The created database system in phpmyadmin :



My solution is to create a Tree(as above shown) with the root being a dir where each node from the root can either be a file or a directory. Also, each node can also have a lot of nodes linked to it, that can either be Directories or Files. I designed a NodeParent class and a NodeChild class for inserting the data from the inputFile.txt. Parent instantiations for the provided sample would be : Documents, Images, Works, Accountant, Program Files, Skype, and Mysql . Child objects would be the rest of the files. Each NodeParent object should have 2 ArrayLists of NodeParent type elements and NodeChild Type elements, sincere in every directory can have a sub-directory as its child. Hence, the diagrams for the NodeParent class and the NodeChild class are :





- **My first algorithm** for inserting the data from the input file to the database had 2 parts:

1)Creating objects and storing the data as their fields
2)Going through each object and then store their fields in the database

Report about the exercise sheet

1)Creating objects and storing the data as their fields

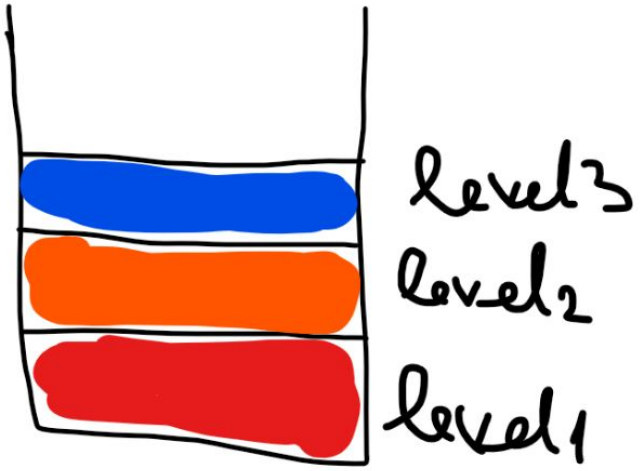
The algorithm:
Whenever we find “<Red>” we will create a new Parent object and will add it in a Stack. The name of the Parent will be the next word after <Red>. If the stack is empty this means that we are not in a Parent object. Whenever the stack is not empty and we find another “<Red>” we create a new Parent object, we peek() to get the top of the stack, we put that new Parent into the ArrayList<Parent>, then we put that new Parent into the stack. Whenever we find “<Blue>” we will create a new Child that has the name equal to the next word after “<Blue>”, peek() to get the top element of the stack and add that Child to the ArrayList<Child> of the top of the stack. Whenever we find “</Red>” we will pop() the top element of the Stack.

2)Going through each object and then store their fields in the database

It will be a recursive traversing tree algorithm.

- **My second algorithm** has only 1 part(Creating the objects and inserting their fields into the database within the loop that reads the .txt file):

Each level of the stack created in the 1) part of the first algorithm will have a NodeParrent object that represents a directory:



For adding the directories:

- The first level of the stack is for the root (e.g. “Documents”) and every NodeParent object that will be in this level will be added in the “C:\” table.
- The second level of the stack is for the root (e.g. “Images”) of the created subtree within the root from the first level and every NodeParent object that will be in this level will be added in the “Directories” table.

- The third level of the stack is for the directories (e.g. “Accountant”) that are within the NodeParent from the second level and every NodeParent object from this level will be added in the “Contents” table, with the boolean flag is_dir set on true.

For adding the files:

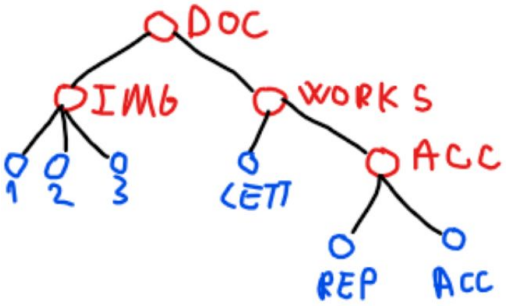
- If we are on the level 2 of the stack, we create a new ChildNode object, we add that child to the “Contents” table and set the boolean flag is_dir to false.
- If we are on the level 3 of the stack, we create a new ChildNode object, we add that child to the “NewTable1” with the boolean flag is_dir set on false.

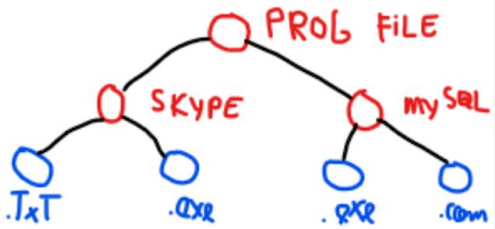
For the one by many relationships, I will use _SESSION[“value”] to store the mysqli->insert_id and the stack to determine where to insert the values into the DB.

Q3.Other possible solutions

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

Solution for Q4.
Now I have an array of 2 NodeParent objects(Documents and Program Files), each being the root of a tree:





I will use the Breadth-first search algorithm to traverse the tree with a custom adaptation. Whenever I see a possible match for the word there are 2 cases:

- If that node is a NodeChild, I print the path from the root to that node.
- If that node is a NodeParent, I print the path from the root to that node.Now I have 3 cases:
- The node NodeParent has no other NodeParent nodes, in this case I iterate through the children of that node, creating a path for each one of them.
- The node NodeParent has no NodeChild nodes and it has only NodeParents Nodes, in this case I will print the whole subtree.
- The node NodeParent has NodeParent nodes and NodeChildren nodes in this case I print all the path of NodeChildren and then I iterate through all the NodeParent nodes and I continue the algorithm.

Report about the exercise sheet