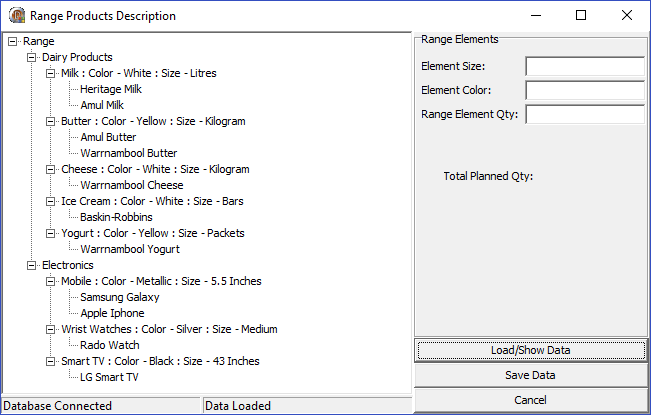
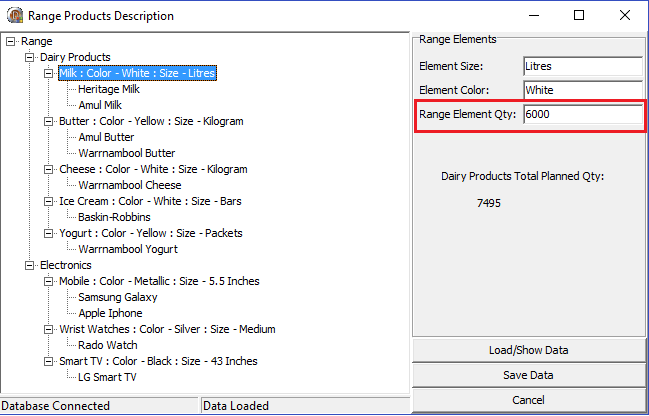
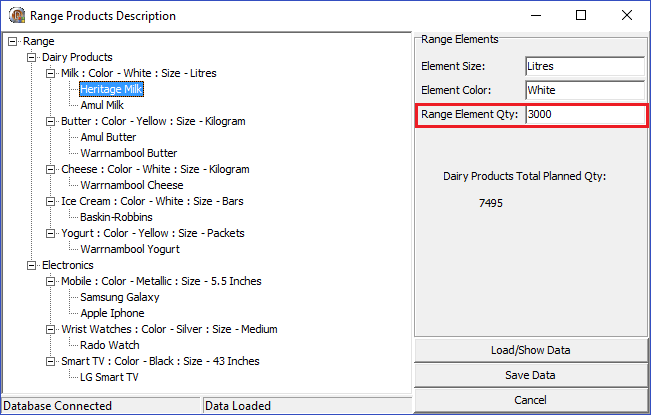
1. Below is the initial view of the application upon Load/Show Data function:



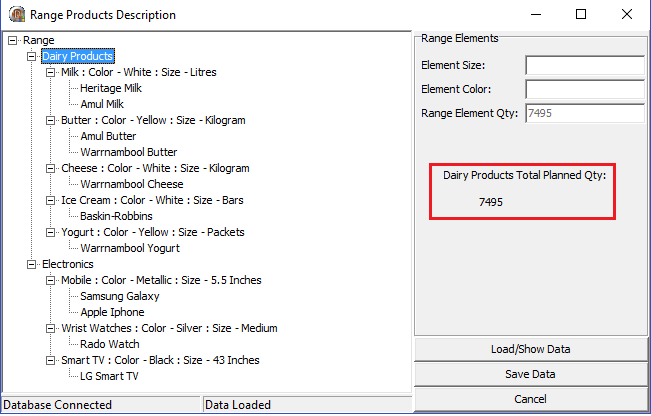
1. It describes the various types of Ranges as three level of tree structure.
2. Root Nodes (‘Dairy Products’, ‘Electronics’ here)
3. Under that the Product element is shown (Milk, Butter, Cheese etc ) with their size and colors appended.
4. Under Range Elements, there are Range Sub Elements which describes the actual product.
5. The application has the functionality implemented for getting the planned Qty for a particular Element of a Range as shown below (highlighted in red):



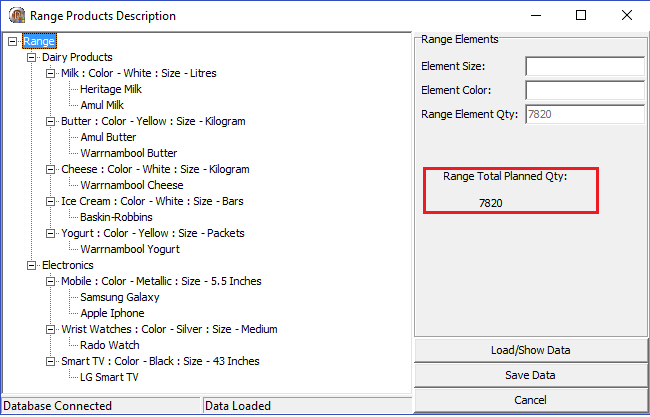
1. Also we can see the individual qty of a particular sub-element by selecting that particular node as below (highlighted in red):



1. Also we can get the particular range total qty upon selecting that Node. Suppose ‘Dairy Products ’ is selected, then qty will be as shown below (highlighted in red)



1. And when we select the Root Node i.e. ‘Range’, we can see the total Range qty across all the ranges as shown below (highlighted in red)



1. Code has been designed simple and to maximum extent, elegant.
2. The application is designed in such a way that it has segregated the concerns listed below:
3. Connecting to database
4. Loading data into data aware component (Used Data Module).
5. Loading the same data into memory with appropriate data structure.
6. Showing the loaded data into visual component on main screen (tree view here).
7. Implementing saving mechanism separately which is intact of UI part.
8. Implementing the methods for getting total planned qty based on Range, element and whole department of range (all ranges).
9. Customized List class to implement my own Tlist with the particular record type for providing a better dependency injection mechanism.
10. Which handles the freeing of memory exclusively for record pointers.
11. We don’t have to typecast everything to use it.
12. The serializer class is well defined for loading and then showing the data onto UI with the help of Observer design pattern based on event handling.
13. Almost tried to use best possible design patterns and most latest Delphi techniques like
14. Adapter design pattern (where TDataSerializer class is acting as an adapter between main screen class and the recList class, data module class).
15. Builder Factory design pattern is followed in the record type by customizing the constructor of the classes.
16. Tried to follow Iterator pattern is followed to traverse tree view nodes.
17. Observer pattern is implemented to load and show the data at one shot without UI interfering into it exclusively for showing the data.
18. Everywhere the data is being logically implemented or modified using the customized list rather than using the query component.
19. Tried using class methods, tree view methods and events exclusively for effective solution.
20. Tried implementing most of the object oriented concepts like encapsulation (by hiding the actual implementation logic in private) as well as by providing properties.
21. Run time and design time polymorphism also implemented at class abstraction level. So that if a user inherits from TDataSerializer (loading and saving mechanism methods as well as all the qty calculation methods), so that they may enhance the functionality of those methods.

DATABASE PART:

The script file is already checked in as a part of the code repository, which defines all the table structures used, their relationships and the demo data used while implementation.

*Please review the above code and its implementation and provide your valuable inputs on it.*

Thanks & Regards,

Gaurav Tripathi