Table Generation Language TabPro

Project Proposal

Rajat Dixit UNI: rd2392
Anureet Dhillon UNI:ad2660
LakshmiNadig UNI:ln2206

1. Introduction/Objective

The objective of this language is to give the user both a flexible and simple way of generating tabulated form of data. This language should aid the user in visualizing the data/information. The user should be able to define the organization of the data in a simple way and at the same time give them the power to manipulate the data before generating the final data in a table layout.

2. Language Features

1. The developer should be able to iterate through lists and perform calculations for each row or column.
2. Provide the developer with filtering support where the final rows displayed will be based on the filtering condition being satisfied.
3. Provide the developer with sorting support where the final rows displayed will be sorted as specified by the columns set to be the sort column.
4. The developer should be able to specify the format of the output – text or HTML. Html would be the default.
   The code snippets in the section below demonstrate the features highlighted above.

3. Language Overview

3.1. Variables, Identifiers, Data Types and Operators

3.1.1. Variables:

An identifier is a sequence of characters and digits; the first character must be alphabetic. Special characters like @, $, etc. are not permitted.

3.1.2. Operators:

The language would support operators “+”, “-“, “*” and “/” for the mathematical operations and [] to access rows and columns.

3.1.3. Data Types

Variables can be declared to be one of the following data types:

- A number data type will be declared using the syntax **num:**
  Eg. set num: point = 5;
- A string data type will be declared as **string:**
  Eg. set string: greet = “Hey”;

- A column data type will be declared using the syntax **col:**
  This data type would be used to declare a list of all values in a particular column.
  Eg. set col: mycol = {1,58,12,17,99};
- A row data type will be declared using the syntax **row:**
  This data type would be used to declare a list of all values in a particular row.
  Eg. set row: myrow = {3,88,45,33,12,77,59,48};

3.2. Control structures

3.2.1. for_each_in(row/col)

This will provide a looping mechanism for all elements in the specified row or the column.

3.2.2. Loop(row/col, start index, end index)

This will loop on a particular row or a column in the specified range of their indexes. For instance, if I want to loop from the fifth index to the ninth index of the given row or column, following code would achieve that:

```
loop( 5,7){
        set mycol = mycol+5;
}
```

3.2.3. loop (condition)

This will provide a looping mechanism as long as the condition holds true.

3.3. User Defined Functions

A user function will be declared as "set function". A function is defined as a function of either a set of columns that returns another column or a function of rows that returns another row. The language engine will then apply this function body to all elements of a row or a column. A function can also be a function taking simple data types int and string ans returns a simple data type. The two cannot be used together – For eg., we cannot have a function that accepts a simple data type and a row or a column. The arguments will be included in the parenthesis following the function name. A colon will follow the closing parenthesis of the argument list and separate the return parameter. The code snippet inside the function would be enclosed in curly braces.

```
set function sqr(col: val) col:answer{
 set answer = val * val;
}
set function product(num:I, num:j) num:k
{
```

```
        k = I * j;
}
```

3.4. System functions and reserved words

3.4.1. col_heading

This is a reserved word; a special list that holds the label values for the column. This is a required field that the programmer has to set.

3.4.2. row_heading

This is a reserved word; a special list that holds the label values for the rows. This is optional and need not be specified by the programmer.

3.4.3. row_limit

This is a reserved word that defines the limit for the row i.e. the maximum number of rows that can be present in the table.

3.4.4. col_limit

This is a reserved word that defines the limit for the column i.e. the maximum number of columns that can be present in the table.

3.4.5. col_sort_index

This is a reserved word; a setting that can be used by the developer to set the column by which the rows need to be sorted.

3.4.6. row_filter_condition

This is a reserved word; a setting that can be used by the developer to set the filtering condition by which the rows will be filtered before final display.

3.4.7. Other reserved keywords(separated by comma):

set, num:, string:, loop, for_each_in, return, function, row, col, filter_condition, sort_col_index
Any line starting with ? is a comment.

4. Example Code

4.1. Example to iterate through a list and fill cells after calculation

Problem: I would like to generate a quick reference card for the unit of distance measurement in both the metric and English system for distances ranging from 0 to 25 miles in increments of 5. I would like to have the output tabulated with Miles and Kilometers as the headings.

Solution Code:

```
set col_heading = “Miles” “Kilometers”;
set  row_size = 5;
set col: miles = {0,5,10,15,20,25} ;
set col_sort_index =  miles;
set function computeKms(col:miles) : kilometers{
     set kilometers = miles * 1.6;
}
```

Solution Output:

Miles	Kms
5	8
10	16
15	24
20	32
25	40

4.2. Example to iterate through a list and fill cells using sort and filter support

Problem: I have the scores for students Tom, Dick and Harry for HW1 and HW2. I would like to tabulate this data along with the third column showing the average score of each student and sort it by the average score. In addition I should have only rows that have an average greater than the score 70

Solution Code:

```
set col_heading = “HW1” “HW2” “AVG”;
set row_heaading= “TOM” “DICK” “HARRY” “TOTALS”;

set  row_size = 4;
set col: HW1= {60, 70, 80} ;
set col: HW2= {66, 76, 86} ;
set filter_condition =  AVG >= 70;
set function computeAvg(col:hw1, col:hw2) : Avg{
     set AVG = miles * 1.6;
}
Set num:sumHw1 = 0;
Set num:sumHw2 = 0;
Set num:sumAvg = 0;
loop(0, 2)
{
sumHw1 = sumHw1+hw1[currIndex];
sumHw2 = sumHw2+hw1[currIndex];
sumAvg = sumAvg +hw1[currIndex];
}
Totals[0] = sumHw1;
Totals[1] = sumHw2
```

Totals[2] = sumAvg

Solution Output:

	HW1	HW2	AVG
DICK	70	76	73
HARRY	80	86	83
TOTAL	150	162	152

5. Library Support

The language will provide one library for some common statistical functions. This will be the "statistics" library which will support a few of the statistical functions like finding mean, median, mode and variance.

6. Nice to Have

1. **File support to read delimited data from a text file:** In the above examples we see that the values in the column are being initialized. This may however be difficult in a practical situation where the user has the data available in the form of a text or CSV file. The language can be enhanced to perform column and row data initializations through text files
2. **PDF Files:** Generate the table data in a PDF form
3. **Templates:** The language can be extended to also include a set of templates. For instance, the code in example is trivial and can be used as a template for all kinds of conversions. Instead of the user/developer writing code to generate the conversion reference card, with the template feature the user would be able to pass an additional template options to the language engine and the engine would automatically generate source code. This source code can then be later either edited for further improvements or directly run for the table generation.
4. **Have a graphical representation of the output data:** The generated table can also be represented in different graph formats as in pie charts, bar graphs, etc.