ColumnarListReport

REPORTNET HOW-TO: FORMATTING A LIST REPORT INTO PAGE-COLUMNS

Wood Foster-Smith
August 16, 2007

Objective:

To create a list report that wraps into columns across the page ("page-columns") before breaking to the next page. This is useful when an ordinary list report has a layout that is narrow (i.e. it’s only a few list columns wide, and more than one of the list could fit across the page) and breaking at the bottom of each page takes up more space than is necessary.

Challenge:

ReportNet does not have settings for list reports that will automatically wrap the list into page-columns.

Solution:

Create a table with one row and as many columns as you want columns across the page. Insert identical lists into each cell, but use the MOD function to select only rows for each list that should appear in that page-column.

Caveats:

This technique will only work for list reports that do not contain group headers and footers.

This technique will only work for list reports where the height of each row is always the same, and the width of each row will never exceed the width of the desired page-column.

Step-by-Step:

1. Create your base list report using a query called “Base.”
   a. Lay out your rows carefully. You must be sure that the each row in your list will always take up the same amount of vertical space on the page. That is, each row must take up one line, or each row must take up two lines, etc. If some rows are one line and some are two (i.e. if the vertical space can vary depending on how much data is in the row, such as when text wraps to a second line), then this technique will not work. Also, you may not use group headers and footers in your list. If you do, this technique will not work.
   b. In the List properties, set the “Contents Height” property to “Minimal” and the “Column Titles” property to “At start of details.”
   c. Add all page headers and footers and adjust the page margins to the way you will want them in your final report. Run the report to PDF format. Make any adjustments to formatting so that your list is the way you want it to look before you proceed further.
   d. Look at the widest row of data in your list and, based on that width, determine how many page-columns will fit across the page. Call this number “pagecolumns”. For this example, pagecolumns = 3.
   e. Count the number of rows that fit on the page in the current layout.
2. Add two calculated fields in the tabular model and query for “Base”:
   a. rownum: running-count([PERSON_UID]) (or any column in the list).
      This will produce a straight running count of every row in the list
      beginning with 1.
   b. mod: mod([rownum], pagecolumns * columnrows) For this example
      the formula is mod([rownum], 3 * 54). The mod function returns the
      remainder after dividing the first argument by the second argument.
      When using [rownum] as the first argument and pagecolumns *
      columnrows as the second argument, what this will result in is a
      straight running count that will count from 1 up through the second
      argument minus 1 (in this case (3 * 54) - 1 or 161), then start over at
      0 and count up to 161 again, and so on until all the rows are counted.
      162 (zero through 161) is the total number of rows that will appear on
      one page (3 page-columns * 54 column rows).
   c. Bring [rownum] and [mod] data elements into the “Base” query.

3. Create copies of “Base” for each page-column you want to create:
   a. In this example, we will have three new queries because we want three
      page-columns. Name them PageCol1, PageCol2, etc.
   b. Into the “Children” area of the tabular model for each query, drag a
      Tabular Reference object, and set the “Query” property for the Tabular
      Reference object to “Base.” This is simply a shortcut to avoid having to
      re-create the same tabular model for each of the new queries.

4. Define each page-column query to return only the rows that should show in
that page-column.
   a. Open the tabular model for PageCol1 and drag a filter object into it.
      Define the filter for [mod] between 1 and columnrows (in this example,
      [mod] between 1 and 54). This will cause only the first 54 rows of each
      set of 162 to be returned by this query.
   b. Open the tabular model for PageCol2 and drag a filter object into it.
      Define the filter for [mod] between columnrows + 1 and columnrows * 2
      (in this example, [mod] between 55 and 108). This will cause only rows
      55 through 108 in each set of 162 to be returned by this query.
   c. Repeat for each additional page-column query, defining a filter that will
      return the only the next set of rows out of the total set for the page.
      Note that for the last page-column query, you will have to include rows
      where [mod] = 0, because the 162nd row will have mod 0. This will be
      the last list row in the last page-column on each page. In this example,
      our third and last page-column query will include the filter [mod] = 0 or
      [mod] between 109 and 161.

5. Set up your report to display identical lists for each page-column, each one
   referencing the query that returns only the correct rows for that page-column.
   a. Drag a table object to your report page that has one row, and as many
      columns as you want page-columns. In this example, the table is one
      row by three columns.
   b. Set the Table object’s “Size & Overflow” property to “Width = 100%”.
      This will equally divide the available space across the page for your
      columns.
   c. Cut the finished list object you created in Step 1 (which currently
      references the query “Base”), and paste it into the first cell of the table.
   d. Copy the same list object and paste it into each of the remaining table
      cells. You now have a table containing identical list objects in each
e. Change the query property of each list object so that it refers to the “TableCol” query that applies to that lists’ page-column. So, for the first list, change the Query property to TableCol1; for the second, to TableCol2; etc.

6. You’re done.
   a. In this example, for each set of 162 rows, page-column 1 will display rows 1 through 55; page-column 2 will display rows 56 through 108; and page-column 3 will display rows 109 through 162. The sequence will start over again on the next page: The 163rd row has \([\text{mod}] = 1\), so it will be the first row in the first column on page 2; the 217th row will have \([\text{mod}] = 55\), so it will be the first row in the second column on the page 2; and the 271st row will have \([\text{mod}] = 109\), so it will be the first row in the third column on page 2; and so on for all remaining rows and pages in the report.
   b. Understanding this helps you to understand why the number of columns per page and the number of rows per column must be pre-determined, and must not vary throughout the report; otherwise these formulas cannot work consistently for every column and every page, and you will wind up either repeating rows between columns, or omitting rows completely.