Horizontally Paginated Reports

Doug Hennig

Horizontally paginated reports allow you to print more columns for a given row that will fit on a page. The report prints as many "horizontal" pages as needed. This technique has many applications, including cross-tabulation reports. In this article, Doug shows how to create horizontally paginated reports using the FoxPro Report Designer.

One of the reasons I liked spreadsheet programs from the moment I saw them was their ability to automatically paginate their printed reports both horizontally and vertically. By "horizontally", I mean if your spreadsheet had more columns than could fit on a page, the spreadsheet program automatically printed a page with the same rows as the previous page but with the next set of columns.

Figure 1 represents a bird's eye view of a horizontally paginated report. In this report, there are six columns and six rows but only enough room on each page to print three columns and three rows. Thus, the report consists of two horizontal and two vertical sets of pages, for a total of four physical pages. Although it doesn't matter whether pages are physically numbered horizontally or vertically, I tend to number them horizontally. Thus page 2 is the one with rows A-C and columns 4-6.

Figure 1. Horizontally paginated report.

	1	**2**	**3**		**4**	**5**	**6**
A	x	x	x	**A**	x	x	x
B	x	x	x	**B**	x	x	x
C	x	x	x	**C**	x	x	x
	1	**2**	**3**		**4**	**5**	**6**
D	x	x	x	**D**	x	x	x
E	x	x	x	**E**	x	x	x
F	x	x	x	**F**	x	x	x

I've come across the need to do this many times in FoxPro applications. Two scenarios I worked with immediately come to mind; I'm sure you can think of lots of others.

- *Data-driven columnar reports*: Such a report allows the user to specify which fields to print as columns for the report. For example, a stock market program might print the company name in the first column and each field the user specified for the company in subsequent columns. This can result in a case where there are more columns than will fit on a page, so the report needs to be horizontally paginated.
- *Cross-tabulation reports*: I've created market analysis cross-tab reports where the user could specify two variables (fields in a table of marketing data), one for the rows and one for the columns. The report shows the sales for the intersection of each variable. For example, a cross-tab of products by country shipped to would show product names in the first column, sales to Argentina in the second column, sales to Austria in the third, and so on. Obviously, unless there are only a few countries involved, it's likely this report will need to be paginated horizontally.

Although I've created horizontally paginated reports programmatically (with a lot of effort) in FoxPro for DOS, this approach doesn't give great results in Windows because report positioning is much more difficult to do in Windows. As I'm sure you've discovered, you're much better off using the FoxPro Report Designer to create reports in FoxPro for Windows and Visual FoxPro. However, the Report Designer doesn't horizontally paginate reports. Even worse, while it can print multiple rows (one or more lines per record), it expects you to pre-define the information for each row. Thus, you can't have a variable number of columns for a given report.

The Solution

A few months ago, the need to create reports with variable numbers of columns in Visual FoxPro forced me to revisit the idea of doing horizontally paginated reports using the Report Designer. This article documents the solution I came up with. While it's a little complex, the good news is that it's possible to create a generic solution that can be applied in lots of different situations.

First, some background. In order to create a horizontally paginated report, you need a source of data for the rows, for the columns, and for the cells formed by the intersection of rows and columns. There could be more than one piece of information in a cell; for example, you might have a sales value and the percentage that value is of a row or column total. It's obvious the report's rows come from records in a table just as they do for other types of reports. Columns come from fields in a table (not necessarily the current table), such as product name and country name in the product sales by country example mentioned earlier. The cells at the intersection of rows and columns could be a field or a user-defined function (UDF) that calculates the appropriate value.

Next, because it's easier to understand horizontally paginated reports from a particular viewpoint, let's look at the use of these reports for cross-tabs. Although FoxPro comes with a program (GENXTAB.PRG in FoxPro 2.x and VFPXTAB.PRG in VFP) that does cross-tabs, I prefer not to use it because it creates a table with one record per cross-tab row (products in the case of the product-country cross-tab mentioned earlier) and one field for each column in the cross-tab (country in this example), with the value of each field being the cell value (the intersection of a given row and column). I prefer a cross-tab cursor with one record per cell. This cursor has three fields: the value for the row (the product name), the value for the column (the country name), and the value of the cell (the sales for that product in that country). A cross-tab cursor with this structure is easy to create using a single SQL SELECT statement. For example, using the sample data that comes with VFP, here's a single (albeit long) command that creates a cross-tab cursor of product names, country names, and quantity of each product shipped to each country:

```
select PRODUCTS.ENG_NAME, ;
    CUSTOMER.COUNTRY, ;
    sum(ORDITEMS.QUANTITY) ;
  from CUSTOMER, ;
    ORDERS, ;
    ORDITEMS, ;
    PRODUCTS ;
  where CUSTOMER.CUST_ID = ORDERS.CUST_ID and ;
    ORDERS.ORDER_ID = ORDITEMS.ORDER_ID and ;
    ORDITEMS.PRODUCT_ID = PRODUCTS.PRODUCT_ID ;
  group by 1, 2 ;
  into cursor XTAB
```

The first field in this cursor is the row values (product names), the second is the column values (country names), and the third is the cell values (number of products shipped to the country).

Of course, the downside of this relatively simple SELECT is that it doesn't show the complete picture: it doesn't include products that have never been sold nor countries in which no customer ever bought any product. You need to do a outer join (using the new OUTER JOIN syntax in VFP 5 or using a UNION SELECT clause in earlier version) with both CUSTOMER and PRODUCTS to include these 0-valued records. See TESTHREP.PRG on the Developer Disk for a complete SELECT statement that includes these records.

Now that we've looked at the background issues, here's the logic for printing a horizontally paginated report:

- We'll work with sets of vertical pages, each consisting of as many horizontal pages as necessary. For example, in Figure 1, there are two vertical pages sets, each with two horizontal pages.
- Calculate the number of horizontal pages in each vertical page set by taking the total number of columns to print and dividing it by the number of columns per page.
- At the start of the first horizontal page in a particular vertical page set, save the key value for the current record being printed. We'll need to return to this same record at the start of each horizontal page in the current vertical page set.
- At the start of subsequent horizontal pages in the same vertical page set, SEEK the saved key value so we can print more columns from the same records as the previous horizontal page.
- Because we may have more columns in the FRX than are needed for the last horizontal page in a vertical page set, blanks should be printed for the column headings and cell values for unused columns.
- At the end of a page, decide if we're done with this vertical page set by checking if the current horizontal page matches the number of horizontal pages in each vertical page set. If so, check if there are any more records to print. If not, the report is done. If so, start the next vertical page set at the next record.
- Because we may run out of records (we're on the last vertical page set) before we've printed all the horizontal pages, we need to ensure the report doesn't stop until all horizontal pages have been printed.

OK, so how do we do all this with an FRX that doesn't permit a variable number of columns? Using UDFs, of course. We'll create an FRX with a fixed number of columns (as many as we can jam across the page) and several UDFs to do the hard work, each called as the expression in a field in the FRX (see HPAGE.FRX on the Developer Disk for an example of an FRX with these UDF calls):

- NewPage() is the expression for a "dummy" field in the Page Header band. This UDF calculates the horizontal page number (either incrementing the former one or setting it back to 1 if we just did the last horizontal page in a vertical page set). It also either saves the key value for the current record if this is the first horizontal page or SEEKs the saved key value if not.
- GetColumn() is the expression for each column header in the Page Header band because each horizontal page has a different set of columns. This routine returns the text to print for the header of a given column on a given horizontal page.
- GetCell() is the expression for each column in the Detail band. It returns the appropriate field from the current record based on the column it appears in and the current horizontal page number.
- NewRow() is the expression for a "dummy" field at the right edge of the Detail band. It's main job is to position the table being printed in the report to the next record. While this may seem unnecessary (the report automatically does a SKIP at the end of the Detail band), this is necessary for cross-tab reports where the cross-tab cursor consists of one record for each cell in the report. It's also important because we may run out of records (we're on the last vertical page set) before we've printed all the horizontal pages. Left to its own devices, the FRX would sense that the report is done, so we'd only get the first horizontal page for the last vertical page set.

I created a "driver" program (provided on the Developer Disk) called XTABREP.PRG that works in both FoxPro 2.x and VFP. XTABREP includes the UDFs listed above so everything is in one place. It expects a cross-tab cursor in the current work area and accepts several parameters:

- the heading to print for the first column (for example, "Product Name")
- the header to print on each page (such as "Product Sales by Country")
- the output device (if it isn't passed, "PREVIEW" is used)
- the left margin for the report (if it isn't passed, 0 is used)

In addition to the records in the cross-tab cursor, if we want row and column totals, we'll need records that have these totals. There'll be one record for each row total (the first field contains the product name, the second field is blank, and the third field is the total quantity for the specified product for all countries)

and one record for each column total (the first field is blank, the second field is the country name, and the third field is the total quantity of all products for the specified country). XTABREP takes care of the work of calculating row and column totals so the program calling it doesn't have to bother.

While I was testing XTABREP.PRG, I thought it'd be really nice to allow the user to specify the paper size (such as letter or legal) and orientation (portrait or landscape). However, because these settings affect the number of rows and columns that can fit on a page, I realized I'd need to create a different FRX for each page size and orientation combination. Since I'd previously done a lot of work with creating FRX files programmatically, it became obvious that the ideal solution is to generate a temporary FRX based on the chosen settings and use it for the report. Creating an FRX programmatically is beyond the scope of this article; see XTABREP.PRG if you're interested in how it's done.

The code for XTABREP.PRG is below. This listing doesn't include the CreateReport routine or any subroutines it calls because of the length of the code. See the copy of XTABREP.PRG on the Developer Disk for a complete listing.

```
parameters tcColumn, ;
  tcHeader, ;
  tcOutput, ;
  tnMargin
private lcOutput, ;
  lnMargin, ;
  lcField1, ;
  lcField2, ;
  lcField3, ;
  lcAlias, ;
  lnTotColumns, ;
  lnCurrSelect, ;
  lcTotals, ;
  lcXTab, ;
  lcIndex, ;
  lcKey, ;
  lcReport, ;
  lnWidth1, ;
  lnWidth2, ;
  lnCurrAscii

* These variables need to be defined as PRIVATE even
* in VFP because some subroutines use these values.

private pcHeader, ;
  pcColumn, ;
  paColumns, ;
  pnColsPerPage, ;
  pnTotalHPages, ;
  pcVertKey, ;
  plDoneVert, ;
  pnHPage

#define cnMAX_COL_WIDTH 20
#define ccTOTALS        'Totals'

* If the output device isn't specified, use "PREVIEW".
* Otherwise, use the appropriate output clause for
* REPORT FORM.

do case
  case type('tcOutput') <> 'C' or empty(tcOutput) or ;
    upper(tcOutput) = 'PREVIEW'
    lcOutput = 'PREVIEW'
  case upper(tcOutput) = 'PRINT'
    lcOutput = 'TO PRINT NOCONSOLE'
  otherwise
    lcOutput = 'TO FILE ' + tcOutput + ;
      iif('Visual' $ version(), ' ASCII', '') + ;
      ' NOCONSOLE'
endcase

* If the report header and first column heading aren't
```

```
* specified, use blank values instead. Use 0 for the
* left margin if it isn't specified.

pcHeader = iif(type('tcHeader') = 'C', tcHeader, '')
pcColumn = iif(type('tcColumn') = 'C', tcColumn, '')
lnMargin = iif(type('tnMargin') = 'N', tnMargin, 0)

* Create an array of headings to print in horizontal
* columns (ignore blank entries since they might be
* there because of records in the first column that
* didn't have any record for the second column in the
* source tables). Determine the total number of columns
* we'll be printing (it'll be the number of unique
* columns plus one for the "totals" column).

lcField1 = field(1)
lcField2 = field(2)
lcField3 = field(3)
lcAlias  = alias()
select &lcField2 ;
  from (lcAlias) ;
  where not empty(&lcField2) ;
  into array paColumns ;
  group by 1
lnTotColumns = _tally + 1
dimension paColumns[lnTotColumns]
paColumns[lnTotColumns] = ccTOTALS

* Create a cursor from the cross-tab table because
* we'll be adding records (ignore records with a blank
* first field since they might be there because of
* records in the second column that didn't have any
* record for the first column in the source tables).

lnCurrSelect = select()
select 0
lcCursor = sys(2015)
lcXTab   = sys(2015)
select &lcField1, ;
    &lcField2, ;
    &lcField3, ;
    '1' as SORT ;
  from (lcAlias) ;
  where not empty(&lcField1) ;
  into cursor (lcCursor)
use (dbf(lcCursor)) again alias (lcXTab) in 0
use in (lcCursor)

* Create totals records for the rows.

lcTotals = sys(2015)
select &lcField1, ;
    ccTOTALS as &lcField2, ;
    sum(&lcField3) as &lcField3, ;
    '1' as SORT ;
  from (lcAlias) ;
  where not empty(&lcField1) ;
  into cursor (lcTotals) ;
  group by 1, 2
select (lcXTab)
append from (dbf(lcTotals))

* Add a blank row that'll appear just above the totals
* row.

insert into (lcXTab) (SORT) values ('8')

* Create totals records for the columns.

select ccTOTALS as &lcField1, ;
    &lcField2, ;
```

```
    sum(&lcField3) as &lcField3, ;
    '9' as SORT ;
  from (lcXTab) ;
  into cursor (lcTotals) ;
  group by 1, 2
select (lcXTab)
append from (dbf(lcTotals))
use in (lcTotals)

* Index the table on the first two columns.

lcIndex = sys(3) + '.IDX'
lcKey   = 'SORT+' + lcField1 + '+' + lcField2
index on &lcKey to (lcIndex) compact

* Create an FRX to use for the report. Define the
* number of horizontal columns per page and the total
* number of horizontal pages needed.

lcReport = sys(3)
lnWidth1 = len(evaluate(lcField1))
calculate max(len(trim(&lcField2))) to lnWidth2
pnColsPerPage = CreateReport(lcReport, lnWidth1, ;
  min(lnWidth2, cnMAX_COL_WIDTH), lcOutput = 'TO FILE', ;
  iif(lnWidth2 > cnMAX_COL_WIDTH, 2, 1), lnMargin)
pnTotalHPages = ceiling(lnTotColumns/pnColsPerPage)

* Define some variables so they exist during the
* report. pcVertKey contains the key for the first
* record on this horizontal page, plDoneVert is a flag
* to indicate when we're done vertical pages, and
* pnHPage is the current horizontal page.

pcVertKey  = ''
plDoneVert = .F.
pnHPage    = 0

* Run the report.

#IF 'Visual' $ version()
lnCurrAscii = _asciicols
_asciicols  = 160
#ENDIF
report form (lcReport) &lcOutput

* Clean up and exit.

#IF 'Visual' $ version()
_asciicols = lnCurrAscii
#ENDIF
use
select (lnCurrSelect)
erase (lcIndex)
erase (lcReport + '.FRX')
erase (lcReport + '.FRT')

******************
function GetColumn
******************

parameters tnColumn
private lnColumn
lnColumn = pnHPage * pnColsPerPage + tnColumn
return iif(lnColumn > alen(paColumns), '', ;
  paColumns[lnColumn])

****************
function GetCell
****************

parameters tnColumn
```

```
private lnColumn, ;
  luReturn, ;
  lnRecno, ;
  lcXTab1, ;
  lcXTab2
lnColumn = pnHPage * pnColsPerPage + tnColumn

* If there is no such column or we're done the vertical
* pages, return a blank value.

if lnColumn > alen(paColumns) or plDoneVert
  luReturn = ''

* Save the current record pointer and get the values
* for the current row and column.

else
  lnRecno = recno()
  lcXTab1 = evaluate(field(1))
  lcXTab2 = paColumns[lnColumn]
  do case

* Sort "8" means the blank row just before the totals
* row, so return a blank value.

    case SORT = '8'
      luReturn = ''

* Find the appropriate record in the cursor (either a
* totals or a regular record) and get the value of its
* "cell" field.

    case seek(iif(trim(lcXTab1) == ccTOTALS, '9', ;
      '1') + lcXTab1 + lcXTab2)
      luReturn = evaluate(field(3))

* We couldn't find an appropriate record, so use a 0
* and return to the former
* record.

    otherwise
      luReturn = 0
      go lnRecno
  endcase
endif lnColumn > alen(paColumns) ...
return luReturn

****************
function NewPage
****************

pnHPage    = ((_pageno - 1) % pnTotalHPages)
plDoneVert = .F.
if pnHPage = 0
  pcVertKey = evaluate(key(val(sys(21))))
else
  seek pcVertKey
endif pnHPage = 0
return ''

***************
function NewRow
***************

private lcField, ;
  lcValue
lcField = field(1)
lcValue = evaluate(lcField)
scan while &lcField = lcValue
endscan while &lcField = lcValue
if eof() and pnHPage + 1 < pnTotalHPages
```

```
  plDoneVert = .T.
  seek pcVertKey
else
  skip -1
endif eof() ...
return ''
```

Here's an example (taken from TESTHREP.PRG on the Developer Disk) of calling XTABREP after creating a cross-tab cursor using a SQL SELECT:

```
do XTABREP with 'Product', 'Product Sales by Country', ;
  'preview'
```

This specifies that the heading for the first column is "Product", the title to print on the top of every page is "Product Sales by Country", the report should be previewed rather than printed, and left margin is the default of 0 because it isn't passed as a parameter.

If you want to see an example of how XTABREP can do its job, run TESTHREP in either FoxPro 2.x or VFP. It uses a set of tables (included on the Developer Disks) taken from the VFP sample data but converted to 2.x format so they can be opened under both FoxPro 2.x and VFP.

Conclusion

Horizontally paginated reports are a wonderful addition to nearly any application. Since cross-tabs are really easy to do using the powerful features of the SQL SELECT command, you can quickly create a dialog form asking the user to select two fields from a table, do the SQL SELECT, and call XTABREP to do the rest of the work. Your users will love it!

Doug Hennig is a partner with Stonefield Systems Group Inc. in Regina, Sask., Canada. He is the author of Stonefield's add-on tools for FoxPro developers, including Stonefield Database Toolkit for Visual FoxPro and Stonefield Data Dictionary for FoxPro 2.x. He is also the author of "The Visual FoxPro Data Dictionary" in Pinnacle Publishing's "The Pros Talk Visual FoxPro" series. Doug has spoken at user groups and regional conferences all over North America. He was a Microsoft Most Valuable Professional (MVP) for 1996. CompuServe 75156,2326.