1010DATA

Year-Over-Year Analysis

Contents

Retail Year-Over-Year (YOY) Analysis

Retailers frequently measure their performance by comparing current year numbers to prior year numbers. This type of analysis is referred to as a year-over-year (YOY) analysis and can be done many ways using basic 1010data functionality. To start, we will examine how to easily perform a YOY calculation using a simple tabulation, some G_Functions, and some basic String Functions.

One of the most basic ways to measure a retailer's trends, whether by store, department, SKU, or any other dimension, is by finding the difference between the current year's performance and the prior year's performance in terms of dollars or percentages. Let's first answer the question: *How much more/less are my sales in a given month as compared to that same month the prior year in terms of dollars?* We will use the Sales Detail table as the basis of our analysis. This type of transactional data is quite common among retailers.

Sales Detail

Columns 1-16 of 16, Rows 1-24 of 3,314,753,767

Trans ID	Date	Time	Store	SKU	Extended Sales	Qty/ Wgt	Promo	Cost	Customer	Department	Group	Division	Sub-Division	Primary Segment	Secondary Segment
-1746773251	12/10/11	00:00:00	199	211611	0.9	1.00	0	0.65	e2c34159	35	311	4	3	high value	perimeter
-1746773251	12/10/11	00:00:00	199	92025	0.1	1.00	0	0.1	e2c34159	35	311	4	3	high value	perimeter
-1746773251	12/10/11	00:00:00	199	49061	1.65	1.00	0	0.92	e2c34159	35	384	4	3	high value	perimeter
-1746770521	12/10/11	00:00:00	70	100226	12.81	1.00	0	10.58		42	23	4	3		
-1746770521	12/10/11	00:00:00	70	335799	0.82	1.00	0	0.82		42	23	4	3		
-1746767474	12/10/11	00:00:00	164	55890	4.52	1.00	0	4.14	2a6d5f19	28	399	4	5	review	
-1746767474	12/10/11	00:00:00	164	398440	1.07	1.00	0	0.8	2a6d5f19	35	86	4	5	review	
-1746767474	12/10/11	00:00:00	164	305295	1.07	1.00	0	0.8	2a6d5f19	35	86	4	5	review	
-1746767474	12/10/11	00:00:00	164	118695	3.59	1.00	0	2.63	2a6d5f19	55	272	4	5	review	
-1746762972	12/10/11	00:00:00	181	217462	11.83	2.00	0	14.74		49	364	4	4		
-1746739362	12/10/11	00:00:00	65	303410	12.51	1.00	0	11.3		4	349	4	4		
-1746739361	12/10/11	00:00:00	65	239320	16.77	2.00	0	16.77	31d7cd04	10	136	4	4	review	
-1746739360	12/10/11	00:00:00	65	499092	1.22	1.00	0	0.99		20	448	4	4		
-1746739360	12/10/11	00:00:00	65	140199	0.08	1.00	0	0.08		20	448	4	4		
-1746736249	12/10/11	00:00:00	138	342130	1.01	2.00	0	0.51	a80e6a5e	35	311	4	4	convenience	
-1746736249	12/10/11	00:00:00	138	92025	0.2	2.00	0	0.2	a80e6a5e	35	311	4	4	convenience	
-1746736249	12/10/11	00:00:00	138	462501	2.28	1.00	0	1.28	a80e6a5e	42	23	4	4	convenience	
-1746736249	12/10/11	00:00:00	138	279697	0.09	1.00	0	0.09	a80e6a5e	42	23	4	4	convenience	
-1746733187	12/10/11	00:00:00	167	244196	3.56	1.00	0	2.36	f2a66aa9	35	494	4	5		
-1746733187	12/10/11	00:00:00	167	74749	1.93	1.00	0	1.02	f2a66aa9	35	351	4	5		
-1746733187	12/10/11	00:00:00	167	277667	1.65	0.79	0	0.56	f2a66aa9	36	65	4	5		
-1746733187	12/10/11	00:00:00	167	247560	1.72	2.39	0	0.97	f2a66aa9	36	468	4	5		
-1746733187	12/10/11	00:00:00	167	15198	0.83	1.00	0	0.2	f2a66aa9	36	65	4	5		
-1746733187	12/10/11	00:00:00	167	104786	3.2	1.00	0	2.43	f2a66aa9	35	86	4	5		

The Sales Detail table contains a transaction number, transaction date, and SKU for every transaction at our retailer. In addition, the above table includes an **Extended Sales** (`xsales`) column indicating the price of each SKU sold. This column will be important for our YOY analyses.

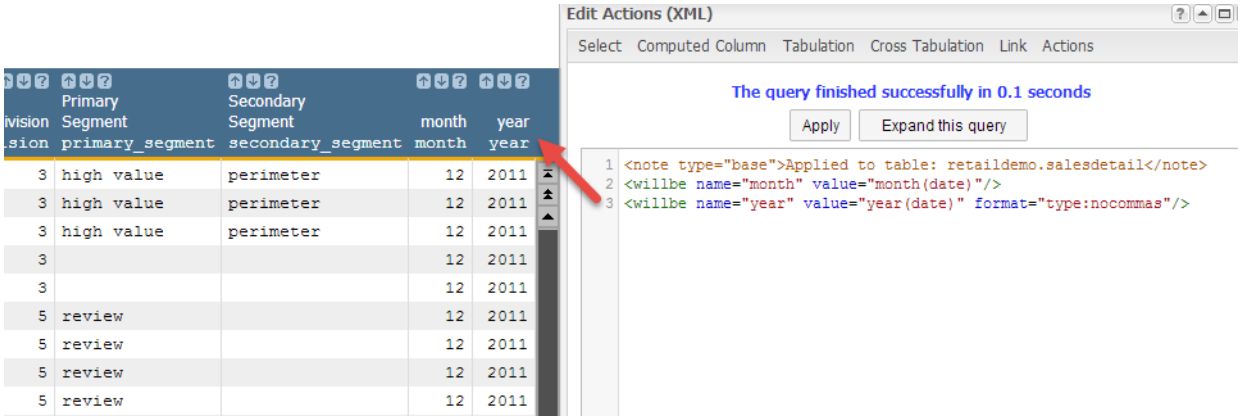
Revisiting our central question – *How are the sales for each month compared to the sales for the same month the previous year?* – we first want to determine what month and year each transaction occurred in. To do this, we can use two simple String Functions to extract the month and year from the **Date** column.

In the 1010data GUI, click **Actions** > **Edit Actions (XML)...** and enter the following text:

```
<note type="base">Applied to table: retaildemo.salesdetail</note>
<willbe name="month" value="month(date)"/>
<willbe name="year" value="year(date)" format="type:nocommas"/>
```

Both the *month(X)* and *year(X)* functions work to extract the relevant information from a date column. Both functions produce an integer as the result: The `month(X)` function produces a one- or two-digit integer representing the month of the date, and the `year(X)` function produces a four-digit integer. (If you prefer using the month name, whether JAN, Jan, January, etc., you can use the *case(X;V1;R1;V2;R2;...;D)* function, which we will touch on at the end of this analysis.) Multiple date formats can be input into these functions, including the standard *MM/DD/YY* format in the table above.

Note: Keep in mind that the `format="type:nocommas"` isn't necessary for the calculations to work correctly, but it will simply make the year *look* correct in the resulting column.



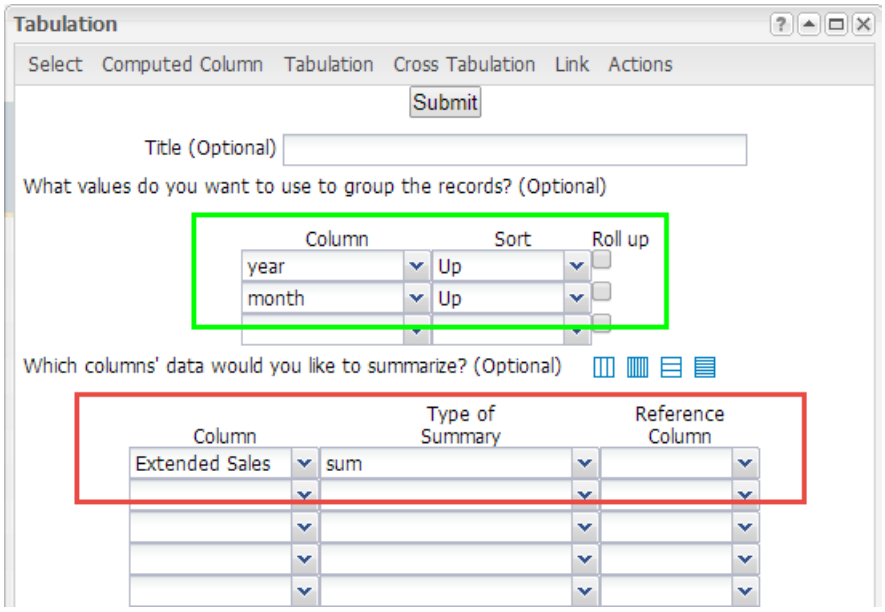
Now that we have all the time information we need, we can begin to aggregate the sales data for each month/year grouping. The simplest way to aggregate the data is by using a tabulation. Because we are concerned with sales by month/year, we will group by both of these new columns and sum up sales.

Here are our requirements for the tabulation:

- Group by the month and year
- Calculate the sum of sales

You can use the 1010data GUI to perform this tabulation. Just go to **Analysis** > **Tabulation...**

Let's take a look at the **Tabulation** dialog to understand how to create the aggregations we're interested in. After, we'll quickly run through the Macro Language code generated by this tabulation. Here's the dialog:



Now that we've defined our grouping metric and the summarizations we're interested in, click **Submit** to get the results:

Tabulation on Sales Detail

Columns 1-3 of 3, Rows 1-33 of 38

year year	month month	Sum of Extended Sales t0
		12,158,276,221.50
2008	11	362,047,434.17
2008	12	369,622,608.06
2009	1	341,267,499.07
2009	2	307,847,842.57
2009	3	334,931,706.04
2009	4	330,741,911.35
2009	5	354,047,851.03
2009	6	332,405,176.11
2009	7	339,477,556.14
2009	8	340,786,203.39
2009	9	325,012,545.24
2009	10	333,035,826.61
2009	11	337,674,632.08
2009	12	354,602,936.23
2010	1	321,249,081.27
2010	2	300,212,383.56

We now have the sales data aggregated by each year/month grouping, as shown above. This makes month-to-month comparisons much cleaner to calculate. Before we move to the next step, let's take a closer look at the 1010data Macro Language produced by the tabulation.

```
<note type="base">Applied to table: retaildemo.salesdetail</note>
<willbe name="month" value="month(date)"/>
<willbe name="year" value="year(date)" format="type:nocommas"/>
<tabu label="Tabulation on Sales Detail" breaks="year,month">
  <break col="year" sort="up"/>
  <break col="month" sort="up"/>
  <tcol source="xsales" fun="sum" label="Sum of`Extended`Sales"/>
</tabu>
```

When a tabulation is created using the **Tabulation** dialog, the default name for the column is `t0` (which you can see at the top of the **Sum of Extended Sales** column in the screenshot above). Let's give it a more meaningful name. In the **Edit Actions (XML)** dialog, add the following text that appears in bold:

```
<note type="base">Applied to table: retaildemo.salesdetail</note>
<willbe name="month" value="month(date)"/>
<willbe name="year" value="year(date)" format="type:nocommas"/>
<tabu label="Tabulation on Sales Detail" breaks="year,month">
  <break col="year" sort="up"/>
  <break col="month" sort="up"/>
  <tcol name="tot_sales" source="xsales" fun="sum" label="Sum
 of`Extended`Sales"/>
</tabu>
```

Note that we grouped by year and then month, and not the other way around. While you can just as easily input the break columns in the reverse order, 1010data will sort the columns last to first. In other words, our tabulation sorted the month column in ascending order, and then sorted the year column in ascending order. This is how we can get the appearance we want. If we input the reverse order for the break columns, the resulting tabulation would be ordered the following way:

Tabulation on Sales Detail

Columns 1-3 of 3, Rows 1-35 of 38

month month	year year	Sum of Extended Sales tot_sales
		12,158,276,221.50
1	2009	341,267,499.07
1	2010	321,249,081.27
1	2011	308,040,557.45
2	2009	307,847,842.57
2	2010	300,212,383.56
2	2011	290,866,206.97
3	2009	334,931,706.04
3	2010	321,119,861.57
3	2011	311,605,344.75
4	2009	330,741,911.35
4	2010	317,894,199.60
4	2011	313,434,699.68
5	2009	354,047,851.03
5	2010	339,251,507.66
5	2011	326,014,146.33

Now that we understand how we got our tabulation results and what they mean, we need to compare each month's aggregated sales number to the same number for one year prior. Let's create a new column that will pull the total sales from one year prior using the G_Function *g_rshift(G;S;O;X;N)*. In the **Edit Actions (XML)** dialog, insert the Macro Language code in bold below to create this column.

```
<note type="base">Applied to table: retaildemo.salesdetail</note>
<willbe name="month" value="month(date)"/>
<willbe name="year" value="year(date)" format="type:nocommas"/>
<tabu label="Tabulation on Sales Detail" breaks="year,month">
  <break col="year" sort="up"/>
  <break col="month" sort="up"/>
  <tcol name="tot_sales" source="xsales" fun="sum" label="Sum
 of`Extended`Sales"/>
</tabu>
<willbe name="ly_sales" value="g_rshift(month;;;tot_sales;-1)" format="dec:2"
 label="Last Year`Sales"/>
```

Running the above Macro Language code should give you the following results:

Tabulation on Sales Detail

Columns 1-4 of 4, Rows 1-35 of 38

year year	month month	Sum of Extended Sales tot_sales	Last Year Sales ly_sales
		12,158,276,221.50	
2008	11	362,047,434.17	
2008	12	369,622,608.06	
2009	1	341,267,499.07	
2009	2	307,847,842.57	
2009	3	334,931,706.04	
2009	4	330,741,911.35	
2009	5	354,047,851.03	
2009	6	332,405,176.11	
2009	7	339,477,556.14	
2009	8	340,786,203.39	
2009	9	325,012,545.24	
2009	10	333,035,826.61	
2009	11	337,674,632.08	362,047,434.17
2009	12	354,602,936.23	369,622,608.06
2010	1	321,249,081.27	341,267,499.07
2010	2	300,212,383.56	307,847,842.57
2010	3	321,119,861.57	334,931,706.04
2010	4	317,894,199.60	330,741,911.35
2010	5	339,251,507.66	354,047,851.03
2010	6	320,547,936.92	332,405,176.11
2010	7	332,634,761.40	339,477,556.14
2010	8	322,816,117.63	340,786,203.39
2010	9	306,803,059.30	325,012,545.24
2010	10	319,253,726.93	333,035,826.61
2010	11	319,101,777.07	337,674,632.08
2010	12	341,522,820.20	354,602,936.23

Now let's quickly take a look at the `g_rshift(month;;;t0;-1)` function and what it calculated. The `G` argument is simply the `month` column, since that is the time period we are interested in using in our YOY calculations. Since we want to include all rows in the calculation and do not need to order the data (since it's already sorted correctly), we can leave the `S` and `O` arguments blank. The `X` argument contains the column we wish to use the function on, which would be our new **Sum of Extended Sales** (`tot_sales`) column. Lastly, the `N` argument is `-1` because we are interested in finding the prior row with the same month value. Because each month only has one row per year and every year has a row for all 12 months, the prior row for each month would be for the previous year. Not surprisingly, if there is no data for the prior year's value, the result in the column will be N/A. As we've already seen, we now have a column (**Last Year Sales**) that pulls the value in the `tot_sales` column for the prior year and copies that value into our new column (`ly_sales`) .

The last step in our analysis is calculating the actual change in sales. Every row now contains both values necessary for the calculation: current sales and prior sales. All we have to do is find the difference. The difference can be represented in several ways, but a simple subtraction or percent calculation is most common. Adding the additional code below will satisfy both methods.

```
<note type="base">Applied to table: retaildemo.salesdetail</note>
<willbe name="month" value="month(date)"/>
<willbe name="year" value="year(date)" format="type:nocommas"/>
<tabu label="Tabulation on Sales Detail" breaks="year,month">
  <break col="year" sort="up"/>
  <break col="month" sort="up"/>
  <tcol name="tot_sales" source="xsales" fun="sum" label="Sum
 of`Extended`Sales"/>
```

```
</tabu>
<willbe name="ly_sales" value="g_rshift(month;;;tot_sales;-1) format="dec:2"
 label="Last Year`Sales"/>
<willbe name="yoy" value="tot_sales-ly_sales" format="dec:2"/>
<willbe name="yoy_perc" value="(tot_sales-ly_sales)/ly_sales"
 format="type:pct;dec:2"/>
```

Our new column, `yoy`, simply subtracts last year's sales total from this year's sales total to give an absolute difference. The column `yoy_perc` converts this same subtraction into a percentage increase/ decrease of last year's sales total. The computed columns will look as they do below:

Tabulation on Sales Detail

Columns 1-6 of 6, Rows 1-35 of 38

year year	month month	Sum of Extended Sales tot_sales	Last Year Sales ly_sales	yoy yoy	yoy_perc yoy_perc
		12,158,276,221.50			
2008	11	362,047,434.17			
2008	12	369,622,608.06			
2009	1	341,267,499.07			
2009	2	307,847,842.57			
2009	3	334,931,706.04			
2009	4	330,741,911.35			
2009	5	354,047,851.03			
2009	6	332,405,176.11			
2009	7	339,477,556.14			
2009	8	340,786,203.39			
2009	9	325,012,545.24			
2009	10	333,035,826.61			
2009	11	337,674,632.08	362,047,434.17	-24,372,802.09	-6.73%
2009	12	354,602,936.23	369,622,608.06	-15,019,671.83	-4.06%
2010	1	321,249,081.27	341,267,499.07	-20,018,417.80	-5.87%
2010	2	300,212,383.56	307,847,842.57	-7,635,459.01	-2.48%
2010	3	321,119,861.57	334,931,706.04	-13,811,844.47	-4.12%
2010	4	317,894,199.60	330,741,911.35	-12,847,711.75	-3.88%
2010	5	339,251,507.66	354,047,851.03	-14,796,343.37	-4.18%
2010	6	320,547,936.92	332,405,176.11	-11,857,239.19	-3.57%
2010	7	332,634,761.40	339,477,556.14	-6,842,794.74	-2.02%
2010	8	322,816,117.63	340,786,203.39	-17,970,085.76	-5.27%
2010	9	306,803,059.30	325,012,545.24	-18,209,485.94	-5.60%
2010	10	319,253,726.93	333,035,826.61	-13,782,099.68	-4.14%
2010	11	319,101,777.07	337,674,632.08	-18,572,855.01	-5.50%
2010	12	341,522,820.20	354,602,936.23	-13,080,116.03	-3.69%

Congratulations! You have just completed a YOY analysis using a tabulation. While this method is very useful if you want to do other G_Function manipulations on the aggregated sales data, you can also chose to do the above analysis using a cross tabulation. By using a cross tabulation, you can get the sales totals for a given month across all years in the same row. We just did something similar using `g_rshift(G;S;O;X;N)` to pull last year's sales into the same row as current year's sales. The cross tabulation will get you there much more quickly with only a few minor adjustments to the Macro Language code.

Below is the Macro Language for performing the YOY analysis using a cross tabulation. Only the portions that appear in bold differ from our Macro Language for the previous YOY analysis.

```
<note type="base">Applied to table: retaildemo.salesdetail</note>
<willbe name="month" value="month(date)"/>
<willbe name="year" value="year(date)" format="type:nocommas"/>
```

```
<tabu label="Tabulation on Sales Detail" breaks="month" cbreaks="year"
 clabels="short">
  <break col="year" sort="up"/>
  <tcol source="xsales" fun="sum" label="Sum of`Extended`Sales"/>
</tabu>
<willbe name="yoy" value="m2-m1" format="dec:2"/>
<willbe name="yoy_perc" value="(m2-m1)/m1" format="type:pct;dec:2"/>
<sort col="month" dir="up"/>
```

Note: You will notice that we no longer need to use the `g_rshift(G;S;O;X;N)` function because the cross tabulation nicely creates a separate column for each year.

Tabulation on Sales Detail

Columns 1-8 of 8, Rows 1-12 of 12

	year	2008	2009	2010	2011		
month month	12,158,276,221.50	731,670,042.23	4031831685.86	3862407233.11	3532367260.30	yoy yoy	yoy_perc yoy_perc
1	970,557,137.79	0.00	341,267,499.07	321,249,081.27	308,040,557.45	-20,018,417.80	-5.87%
2	898,926,433.10	0.00	307,847,842.57	300,212,383.56	290,866,206.97	-7,635,459.01	-2.48%
3	967,656,912.36	0.00	334,931,706.04	321,119,861.57	311,605,344.75	-13,811,844.47	-4.12%
4	962,070,810.63	0.00	330,741,911.35	317,894,199.60	313,434,699.68	-12,847,711.75	-3.88%
5	1,019,313,505.02	0.00	354,047,851.03	339,251,507.66	326,014,146.33	-14,796,343.37	-4.18%
6	962,443,637.60	0.00	332,405,176.11	320,547,936.92	309,490,524.57	-11,857,239.19	-3.57%
7	998,866,487.82	0.00	339,477,556.14	332,634,761.40	326,754,170.28	-6,842,794.74	-2.02%
8	973,678,670.39	0.00	340,786,203.39	322,816,117.63	310,076,349.37	-17,970,085.76	-5.27%
9	931,517,832.20	0.00	325,012,545.24	306,803,059.30	299,702,227.66	-18,209,485.94	-5.60%
10	960,732,607.05	0.00	333,035,826.61	319,253,726.93	308,443,053.51	-13,782,099.68	-4.14%
11	1,327,448,212.61	362,047,434.17	337,674,632.08	319,101,777.07	308,624,369.29	-18,572,855.01	-5.50%
12	1,185,063,974.93	369,622,608.06	354,602,936.23	341,522,820.20	119,315,610.44	-13,080,116.03	-3.69%

Now that you know how to perform a YOY analysis two ways, let's take this knowledge a step further. Next, we will determine which quarter had the highest sales and which quarter saw the largest YOY growth.

Quarter with Highest Sales and Highest Growth

The next step in our YOY analysis of sales data is to determine which quarter had the highest sales and greatest percent growth. We'll start with the tabulation we have already produced, which gives sales totals for each month/year combination:

Tabulation on Sales Detail

Columns 1-3 of 3, Rows 1-33 of 38

year year	month month	Sum of Extended Sales t0
		12,158,276,221.50
2008	11	362,047,434.17
2008	12	369,622,608.06
2009	1	341,267,499.07
2009	2	307,847,842.57
2009	3	334,931,706.04
2009	4	330,741,911.35
2009	5	354,047,851.03
2009	6	332,405,176.11
2009	7	339,477,556.14
2009	8	340,786,203.39
2009	9	325,012,545.24
2009	10	333,035,826.61
2009	11	337,674,632.08
2009	12	354,602,936.23
2010	1	321,249,081.27
2010	2	300,212,383.56

As a reminder, we can obtain the above tabulation using the Macro Language below:

```
<note type="base">Applied to table: retaildemo.salesdetail</note>
<willbe name="month" value="month(date)"/>
<willbe name="year" value="year(date)" format="type:nocommas"/>
<tabu label="Tabulation on Sales Detail" breaks="year,month">
  <break col="year" sort="up"/>
  <break col="month" sort="up"/>
  <tcol name="tot_sales" source="xsales" fun="sum" label="Sum
 of`Extended`Sales"/>
</tabu>
```

To determine which month had the highest sales and greatest percent growth, we only need to perform four basic steps:

1. Create a computed column that assigns a quarter to each month in the table
2. Create a tabulation that groups by quarter and totals sales
3. Create a computed column that calculates percent growth
4. Create computed columns that find the highest sales total and the greatest percent growth

Step 1: Assign a quarter to each month. While we could do this by going to **Columns** > **Create Computed Column...**, we're actually going to do everything in the Macro Language. Start with the code we generated last time and then add the last line in bold:

```
<note type="base">Applied to table: retaildemo.salesdetail</note>
<willbe name="month" value="month(date)"/>
<willbe name="year" value="year(date)" format="type:nocommas"/>
<tabu label="Tabulation on Sales Detail" breaks="year,month">
  <break col="year" sort="up"/>
  <break col="month" sort="up"/>
  <tcol name="tot_sales" source="xsales" fun="sum" label="Sum
 of`Extended`Sales"/>
</tabu>
<willbe name="qtr" value="quarter((year*100)+month)" label="Quarter"/>
```

The line above produces the following results:

Tabulation on Sales Detail

Columns 1-4 of 4, Rows 1-35 of 38

year year	month month	Sum of Extended Sales tot_sales	Quarter qtr
		12,158,276,221.50	
2008	11	362,047,434.17	4
2008	12	369,622,608.06	4
2009	1	341,267,499.07	1
2009	2	307,847,842.57	1
2009	3	334,931,706.04	1
2009	4	330,741,911.35	2
2009	5	354,047,851.03	2
2009	6	332,405,176.11	2
2009	7	339,477,556.14	3
2009	8	340,786,203.39	3
2009	9	325,012,545.24	3
2009	10	333,035,826.61	4
2009	11	337,674,632.08	4
2009	12	354,602,936.23	4
2010	1	321,249,081.27	1
2010	2	300,212,383.56	1

The *quarter(X)* function outputs the quarter number associated with each calendar date (e.g., 1, 2, 3, 4). In order to utilize this function, we had to manipulate and combine our two date columns, `year` and `month`, into an acceptable format that can be recognized by the `quarter(X)` function. One of these formats is *YYYYMM*, which is why we included the logic `quarter((year*100)+month)`. Since our two date columns are both in integer format, we can perform mathematical functions on their values. To get a final number in the *YYYYMM* format, we just had to multiply the `year` by 100 and then add the `month` value. For example, if we were looking at the first row, our formula would translate to (2008*100) + 11, which would result in 200811, our *YYYYMM* format!

Step 2: Perform a tabulation and group by the newly created **Quarter** column by entering the text that appears in bold below:

```
<note type="base">Applied to table: retaildemo.salesdetail</note>
<willbe name="month" value="month(date)"/>
<willbe name="year" value="year(date)" format="type:nocommas"/>
<tabu label="Tabulation on Sales Detail" breaks="year,month">
  <break col="year" sort="up"/>
  <break col="month" sort="up"/>
  <tcol name="tot_sales" source="xsales" fun="sum" label="Sum
 of`Extended`Sales"/>
</tabu>
<willbe name="qtr" value="quarter((year*100)+month)" label="Quarter"/>
<tabu label="Tabulation on Sales Detail" breaks="year,qtr">
  <break col="year" sort="up"/>
  <break col="qtr" sort="up"/>
  <tcol source="tot_sales" fun="sum" name="qtr_sales"
        label="Total`Qtr Sales"/>
</tabu>
```

Now we have a very similar summarization of total sales, but by quarter instead of by month. Note that there are two break columns: `qtr` and `year`. If we only grouped by quarter, for example, it would

aggregate sales across all second quarters across all years. By including the `year` column in the breaks, we are distinguishing between the second quarter in different years.

Tabulation on Sales Detail

Columns 1-3 of 3, Rows 1-13 of 13

year year	Quarter qtr	Total Qtr Sales qtr_sales
		12,158,276,221.50
2008	4	731,670,042.23
2009	1	984,047,047.68
2009	2	1,017,194,938.49
2009	3	1,005,276,304.77
2009	4	1,025,313,394.92
2010	1	942,581,326.40
2010	2	977,693,644.18
2010	3	962,253,938.33
2010	4	979,878,324.20
2011	1	910,512,109.17
2011	2	948,939,370.58
2011	3	936,532,747.31
2011	4	736,383,033.24

Step 3: Create a computed column to calculate percent growth.

Similar to our original analysis, we can use the *g_rshift(G;S;O;X;N)* function to calculate the percent growth by quarter. As an additional challenge, let's see if we can do this calculation in a single computed column. Remember, originally we first created a column to copy the prior year's sales value into the current year's row, and then we created another column to actually do the percentage calculation. This time, we are going to combine these two steps into one. Essentially, we are going to substitute in the `value` portion of our `ly_sales` computed column into the `value` portion of our `yoy_perc` computed column.

```
<note type="base">Applied to table: retaildemo.salesdetail</note>
<willbe name="month" value="month(date)"/>
<willbe name="year" value="year(date)" format="type:nocommas"/>
<tabu label="Tabulation on Sales Detail" breaks="year,month">
  <break col="year" sort="up"/>
  <break col="month" sort="up"/>
  <tcol name="tot_sales" source="xsales" fun="sum" label="Sum
 of`Extended`Sales"/>
</tabu>
<willbe name="qtr" value="quarter((year*100)+month)" label="Quarter"/>
<tabu label="Tabulation on Sales Detail" breaks="year,qtr">
  <break col="year" sort="up"/>
  <break col="qtr" sort="up"/>
  <tcol source="tot_sales" fun="sum" name="qtr_sales"
        label="Total`Qtr Sales"/>
</tabu>
<willbe name="yoy_perc" value="(qtr_sales-(g_rshift(qtr;;;qtr_sales;-1)))/
g_rshift(qtr;;;qtr_sales;-1)" format="type:pct;dec:2"/>
```

The above code will result in the following:

Tabulation on Sales Detail

Columns 1-4 of 4, Rows 1-13 of 13

year year	Quarter qtr	Total Qtr Sales qtr_sales	yoy_perc yoy_perc
		12,158,276,221.50	
2008	4	731,670,042.23	
2009	1	984,047,047.68	
2009	2	1,017,194,938.49	
2009	3	1,005,276,304.77	
2009	4	1,025,313,394.92	40.13%
2010	1	942,581,326.40	-4.21%
2010	2	977,693,644.18	-3.88%
2010	3	962,253,938.33	-4.28%
2010	4	979,878,324.20	-4.43%
2011	1	910,512,109.17	-3.40%
2011	2	948,939,370.58	-2.94%
2011	3	936,532,747.31	-2.67%
2011	4	736,383,033.24	-24.85%

If we wanted to find which quarter had the highest sales or the greatest growth, we could just sort the columns in descending order, but let's say we want to rank the values without changing the order of the table. We can do this by creating some new columns using the *g_rank(G;S;O;X)* function.

Step 4: For the final step, add the code that appears in bold below to rank the `qtr_sales` column and the `yoy_perc` column separately.

```
<note type="base">Applied to table: retaildemo.salesdetail</note>
<willbe name="month" value="month(date)"/>
<willbe name="year" value="year(date)" format="type:nocommas"/>
<tabu label="Tabulation on Sales Detail" breaks="year,month">
  <break col="year" sort="up"/>
  <break col="month" sort="up"/>
  <tcol name="tot_sales" source="xsales" fun="sum" label="Sum
 of`Extended`Sales"/>
</tabu>
<willbe name="qtr" value="quarter((year*100)+month)" label="Quarter"/>
<tabu label="Tabulation on Sales Detail" breaks="year,qtr">
  <break col="year" sort="up"/>
  <break col="qtr" sort="up"/>
  <tcol source="tot_sales" fun="sum" name="qtr_sales"
        label="Total`Qtr Sales"/>
</tabu>
<willbe name="yoy_perc" value="(qtr_sales-(g_rshift(qtr;;;qtr_sales;-1)))/
g_rshift(qtr;;;qtr_sales;-1)" format="type:pct;dec:2"/>
<willbe name="rank_sales" value="g_rank(;;;qtr_sales)" label="Sales`Rank"/>
<willbe name="rank_perc" value="g_rank(;;;yoy_perc)" label="Growth`Rank"/>
```

Now you should see two new columns containing integers ranging between 1 and the number of total rows in the table, which in this case is 13. The `g_rank(G;S;O;X)` function will rank the largest value as 1 and then incrementally number each row in descending order.

Tabulation on Sales Detail

Columns 1-6 of 6, Rows 1-13 of 13

year year	Quarter qtr	Total Qtr Sales qtr_sales	yoy_perc yoy_perc	Sales Rank rank_sales	Growth Rank rank_perc
		12,158,276,221.50			
2008	4	731,670,042.23		13	
2009	1	984,047,047.68		4	
2009	2	1,017,194,938.49		2	
2009	3	1,005,276,304.77		3	
2009	4	1,025,313,394.92	40.13%	1	1
2010	1	942,581,326.40	-4.21%	9	6
2010	2	977,693,644.18	-3.88%	6	5
2010	3	962,253,938.33	-4.28%	7	7
2010	4	979,878,324.20	-4.43%	5	8
2011	1	910,512,109.17	-3.40%	11	4
2011	2	948,939,370.58	-2.94%	8	3
2011	3	936,532,747.31	-2.67%	10	2
2011	4	736,383,033.24	-24.85%	12	9

The data clearly shows that the fourth quarter of 2009 is both the highest selling quarter and the highest growth quarter. You just completed a second analysis by regrouping on a different time frame!

If you're interested in taking this process one step further to discover what drives these results, here are some questions you might ask:

- Do higher priced items account for the higher percentage of total sales during the better performing quarters as opposed to the poorly performing quarters?
- Does volume sold account for the higher percentage of total sales during the better performing quarters as opposed to the poorly performing quarters?
- How has the distribution of item price and volume sold changed over time? Does either move in line with the sales trends?

How to convert integers into month names using `case(X;V1;R1;V2;R2;...;D)`

The *`case(X;V1;R1;V2;R2;...;D)`* function allows us to programmatically write, "For column `X`, if the value is `V1`, write `R1`; if the value is `V2`, write `R2`; if the value is `V3`, write `R3`..., otherwise, write `D`." Knowing this, you can now see how useful this function is in our month-naming use case. To convert our original `month` column of integers into a new column containing month names, the only logic we need is:

```
<willbe name="month_name" value="case(month;1;'Jan';2;'Feb';3;'Mar';
                                      4;'Apr';5;'May';6;'Jun';
                                      7;'Jul';8;'Aug';9;'Sep';
                                      10;'Oct';11;'Nov';'Dec')"/>
```

In the above example, we chose to use the three-lettered month abbreviations, but you can write whatever you prefer. Now, you should have a `month_name` column that corresponds to your `month` column in the following manner:

month	month_name
1	Jan
2	Feb
3	Mar
4	Apr
5	May
6	Jun
7	Jul
8	Aug
9	Sep
10	Oct
11	Nov
12	Dec

As you can probably guess, the `case(X;V1;R1;V2;R2;...;D)` function can be incredibly useful in various scenarios. Creating your new `month_name` column is just the first!