1010DATA

G_Functions - A Beginner's Guide

(212) 405.1010 | info@1010data.com | Follow: @1010data | www.1010data.com

Contents

Introduction

In this introduction to 1010data's Group Functions, or as we simply call them: *G_Functions*, we will review the basics of data aggregation, look at examples of conventional methods of aggregation, and learn about the anatomy of all basic G_Functions.

1010data's in-memory, columnar system architecture gives it many unique advantages for data analysis versus other types of databases. And nowhere in the system are these advantages more apparent than in our assortment of Group Functions. These functions provide a powerful methodology for data summaries and aggregation that has some real advantages over more conventional methods such as tabulations.

Data Aggregation

If you are new to the world of Big Data, you may still be familiar with the concept of data aggregation, but you might not be aware of your own familiarity. In essence, aggregation is what we call the process of starting with many data points and arriving at a smaller number of more meaningful data points.

A basic example many people are familiar with is the MS Excel concept of a "pivot table." MS Excel limits the total number of rows to 1 million in a worksheet, so if you ask us, it isn't exactly what we consider Big Data.

In 1010data, we have a very similar concept called a tabulation. Tabulations are a very common way to summarize your data to understand something specific about it. To start, let's look at the most basic example we could think of. We're going to start with a small data table of retail sales transactions. This is what it looks like:

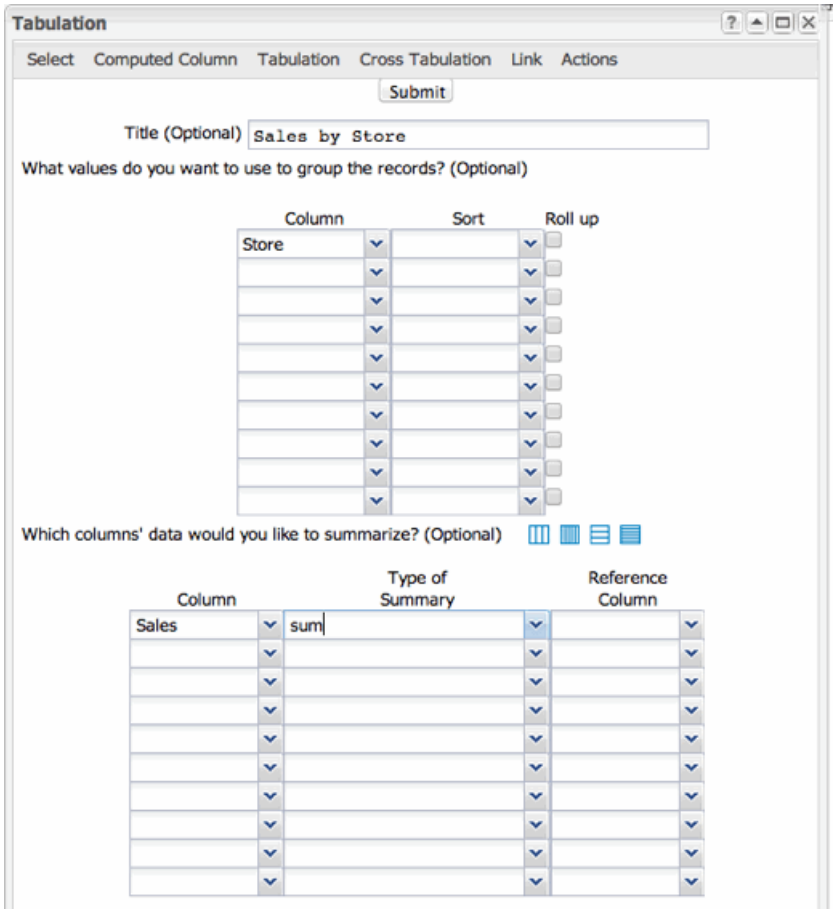
Sales Item Detail

Columns 1-8 of 8, Rows 1-26 of 35

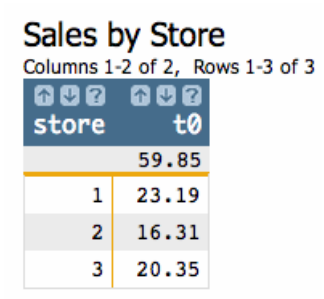
transid	account	store	date	sku	units	sales	cost
531	957	1	05/15/12	366	-1	-5	-1.84
532	478	1	05/15/12	98A	1	0.5	0.25
532	478	1	05/15/12	3B7	1	1.1	0.56
534	738	1	05/16/12	A96	2	6	2.9
534	738	1	05/16/12	65B	1	2.25	1.35
535	709	2	05/15/12	CB7	1	1.65	1.1
535	709	2	05/15/12	96A	1	1.1	1
535	709	2	05/15/12	969	1	1.1	1
536	748	2	05/17/12	3A4	3	1.02	0.39
536	748	2	05/17/12	366	1	5	1.8
537	523	3	05/15/12	CB7	1	1.65	1.1
537	523	3	05/15/12	A96	2	6	2.9
537	523	3	05/15/12	98A	1	0.5	0.25
537	523	3	05/15/12	96A	1	1.1	1
537	523	3	05/15/12	3B7	1	1.1	0.56
538	668	1	05/18/12	CB7	1	1.65	1.1
538	668	1	05/18/12	98A	4	2	1
538	668	1	05/18/12	96A	1	1.1	1
538	668	1	05/18/12	969	1	1.1	1
538	668	1	05/18/12	3B7	1	1.1	0.56
539	25	2	06/19/12	65B	1	5	3.35
540	361	2	06/19/12	3B7	1	1.1	0.56
540	361	2	05/19/12	3A4	1	0.34	0.13
541	469	3	05/22/12	A96	1	4	1.45
541	469	3	05/22/12	98A	1	0.5	0.25
541	469	3	05/22/12	96A	1	1.1	1

Above we have 35 rows of data, organized as follows: 1 line for each item in a single transaction. So, for instance, in transaction 532, two items were purchased, so we have two rows of data. Other data points include the store number, the date and the customer account under which each purchase was made.

We would like to find out the total number of sales for each store (as indicated by the columns outlined in red). This is fairly easy to do with a tabulation. Using 1010data's web-based interface we can easily summarize our data as sales by store. In other words, if we *group* the sales data so that only data for a given store is collected within the group, then we can do whatever we want to that data and it will tell us something meaningful about the store itself. In this case, we merely want to add up all the sales data we've grouped by store. We'll simply go to **Analysis** > **Tabulation...** and set up our tabulation in the **Tabulation** dialog. The setup looks like this:



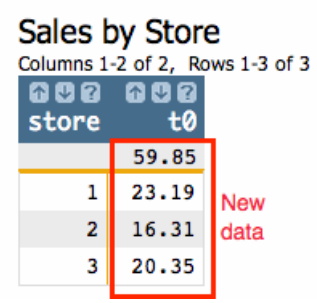
And the result of our tabulation looks like this:



If this looks familiar to you, you're on the right track. We performed this exact same tabulation in our award winning tutorial: "Summarizing Data in 1010data: Tabulation." Follow the link to review that content.

The analysis we just performed on our base table should be fairly apparent whether you're familiar with our earlier tutorial or not. We simply added up the sales figures for each store and then placed those totals in a column. We can now easily see how much in total sales each store had in our data set. Another important thing to note about this operation is that we wind up with some of the original data and some new data. The

store numbers existed in our original table. The sales totals did not. So by producing the sales totals we have added new value to the original information of the store numbers.



Summarizing your data in this way is very useful and if you're coming from the Excel world it can be comforting. However, in 1010data, tabulations are only one way to arrive at useful data summaries. Tabulation, and its exotic cousin the cross-tabulation, provide certain options and advantages. However, due to its design, that being a *columnar database*, we can leverage the 1010data's system architecture to summarize our data while preserving a lot of the information that tabulations eschew. The tools we use for this are called *G_Functions*.

Before we move forward, you should be familiar with using functions in 1010data. Typically, our users will use functions in Macro Language queries or in Value Expressions. This article will use examples in Value Expressions.

Summarizing Data with G_Functions

We can use a single G_Function to perform the exact same calculations as in our tabulation example.

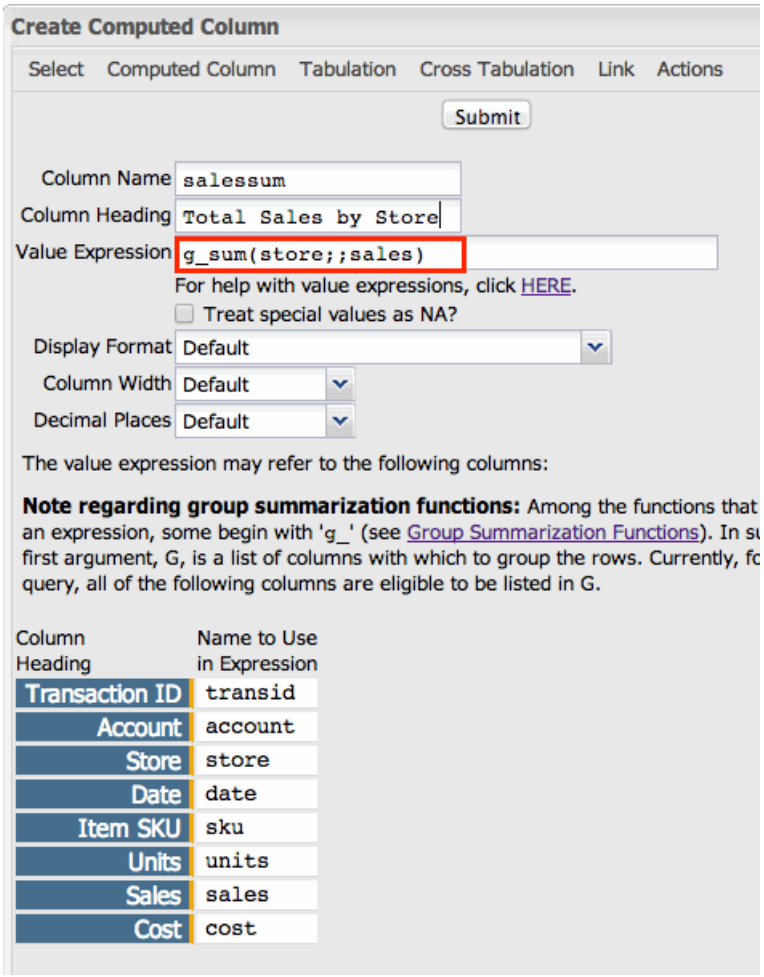
Since we only require the sums of sales for each store, we can use a single G_Function, `g_sum(G;S;X)`. `g_sum(G;S;X)` is one of the most basic G_Functions available in our function library. The three parameters it calls for are available in every single G_Function. While the `S` parameter is very important, we're going to omit it from our first example. Instead, we will simply provide `G` and `X` in order to replicate our tabulation results as closely as possible. In all G_Functions, `G` and `X` represent the following:

- G - is the column name that we will *group* our results by. In this case we are still going to use the `store` column, as we did in our tabulation.
- X - is the column name that we want our G_Function to act on. In this case we are still going to use the `sales` column.

Providing the G_Function with the parameters described above will provide a total sales figure for each store in our table. Here's what the function call looks like in a Value Expression:

```
g_sum(store;;sales)
```

Notice that we are omitting the `S` parameter, but still must separate it from the other parameters with semi-colons. In order to actually apply this Value Expression to the table, we will need to create a computed column. Go to **Columns** > **Create Computed Column...** and enter the Value Expression as shown in the screenshot below:



Once you click **Submit**, you will have a new column in your table that contains the total for each store, for each record from that store:

Sales Item Detail

Columns 1-9 of 9, Rows 1-28 of 35

Transaction ID	Account	Store	Date	Item SKU	Units	Sales	Cost	Total Sales by Store
531	957	1	05/15/12	366	-1	-5	-1.84	23.19
532	478	1	05/15/12	98A	1	0.5	0.25	23.19
532	478	1	05/15/12	3B7	1	1.1	0.56	23.19
534	738	1	05/16/12	A96	2	6	2.9	23.19
534	738	1	05/16/12	65B	1	2.25	1.35	23.19
535	709	2	05/15/12	CB7	1	1.65	1.1	16.31
535	709	2	05/15/12	96A	1	1.1	1	16.31
535	709	2	05/15/12	969	1	1.1	1	16.31
536	748	2	05/17/12	3A4	3	1.02	0.39	16.31
536	748	2	05/17/12	366	1	5	1.8	16.31
537	523	3	05/15/12	CB7	1	1.65	1.1	20.35
537	523	3	05/15/12	A96	2	6	2.9	20.35

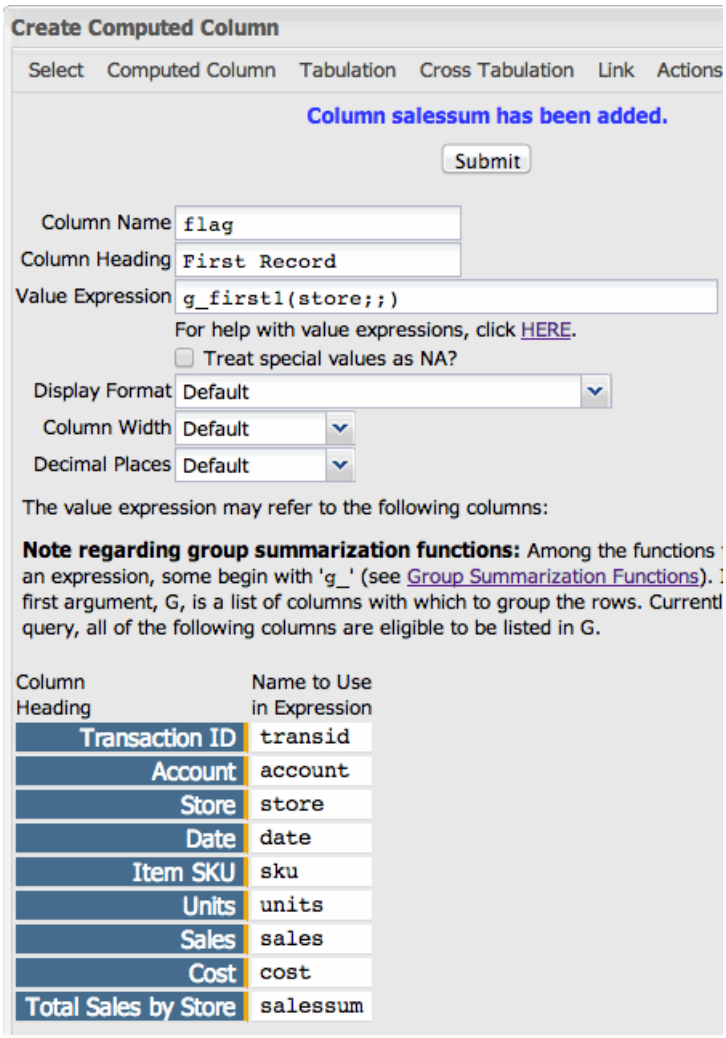
You should immediately notice two main differences here from the results of our tabulation. First, we still have all the information in the table we started with. This is a real advantage of G_Functions. They give you the ability of being able to see your summarization data while maintaining the granularity of your original table. This can be extremely valuable for calculations where each result in the new column might be different. However, in this example, we do need to see the total sales for the store for every single row. Which brings us to the second detail you may have noticed by now: g_functions don't re-format your data.

Depending on how you use G_Functions, a little extra effort to format your results may be required. However, G_Functions can also help us to this end. Since we only want to see one result for each store in our table, it would be helpful to be able to select the rows we want. However, we don't have a unique metric with which to do this. So we need to create one.

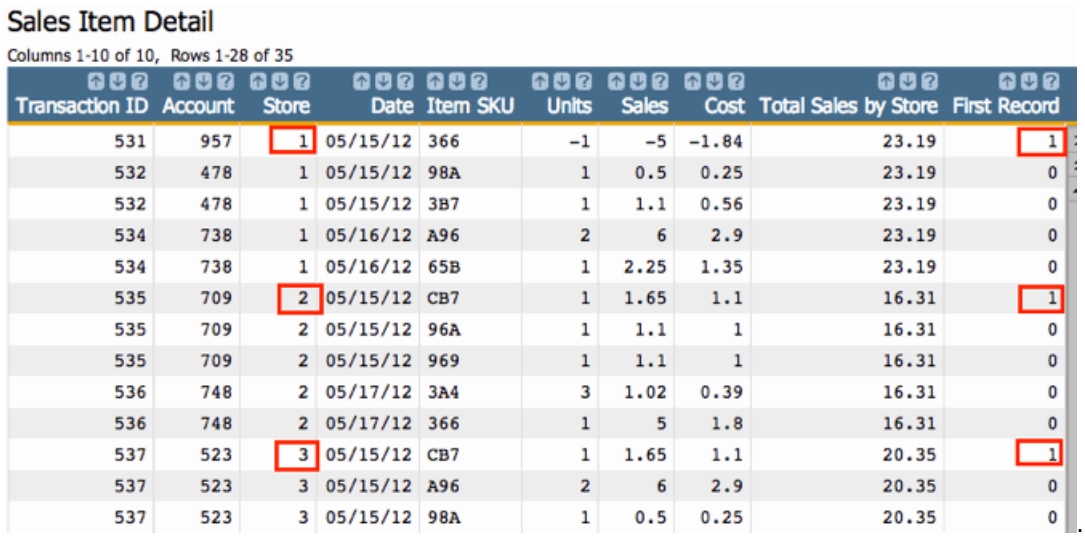
The next step here is to create a *Selection Column*. A *Selection Column* is simply a new column of data that contains either a **1** if the condition for that row is true or **0** if the condition is false. Even better, we can use another G_Function to create our column. In this case, we're going to use the `g_first1(G;S;O)` function. We'll only use the first parameter to produce our column. Go to **Columns** > **Create Computed Column...** and enter the following Value Expression:

```
g_first1(store;;)
```

as shown in the screenshot below:



This will add another column to our table. Each time a store appears for the *first time* in the table, the corresponding value will be **1**. If it isn't the first time a record for a given store has appeared, the corresponding value in the Selection Column will be 0. Let's see the results:

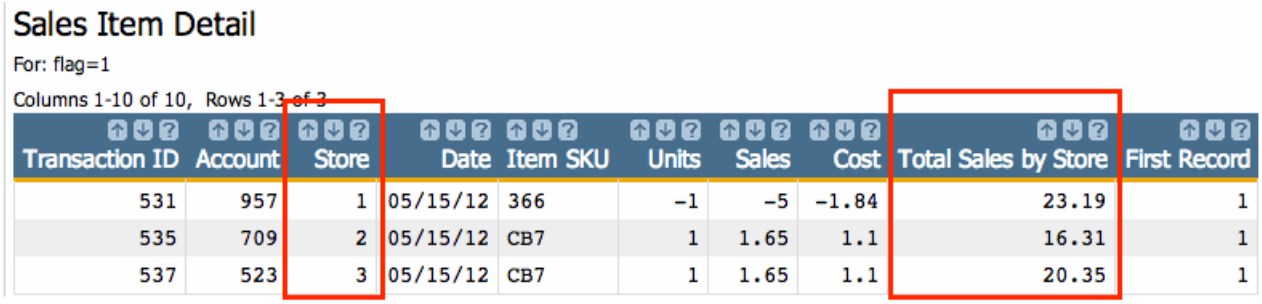


Sales Item Detail

Columns 1-10 of 10, Rows 1-28 of 35

Transaction ID	Account	Store	Date	Item SKU	Units	Sales	Cost	Total Sales by Store	First Record
531	957	1	05/15/12	366	-1	-5	-1.84	23.19	1
532	478	1	05/15/12	98A	1	0.5	0.25	23.19	0
532	478	1	05/15/12	3B7	1	1.1	0.56	23.19	0
534	738	1	05/16/12	A96	2	6	2.9	23.19	0
534	738	1	05/16/12	65B	1	2.25	1.35	23.19	0
535	709	2	05/15/12	CB7	1	1.65	1.1	16.31	1
535	709	2	05/15/12	96A	1	1.1	1	16.31	0
535	709	2	05/15/12	969	1	1.1	1	16.31	0
536	748	2	05/17/12	3A4	3	1.02	0.39	16.31	0
536	748	2	05/17/12	366	1	5	1.8	16.31	0
537	523	3	05/15/12	CB7	1	1.65	1.1	20.35	1
537	523	3	05/15/12	A96	2	6	2.9	20.35	0
537	523	3	05/15/12	98A	1	0.5	0.25	20.35	0

Now we're getting somewhere. Whereas before, we had no way to select rows to reduce our results to the minimum requirement, we can now do exactly that. Go to **Rows** > **Select Rows...** and select on: `flag=1`. By selecting this way, we will have the following results:



Sales Item Detail

For: flag=1

Columns 1-10 of 10, Rows 1-3 of 3

Transaction ID	Account	Store	Date	Item SKU	Units	Sales	Cost	Total Sales by Store	First Record
531	957	1	05/15/12	366	-1	-5	-1.84	23.19	1
535	709	2	05/15/12	CB7	1	1.65	1.1	16.31	1
537	523	3	05/15/12	CB7	1	1.65	1.1	20.35	1

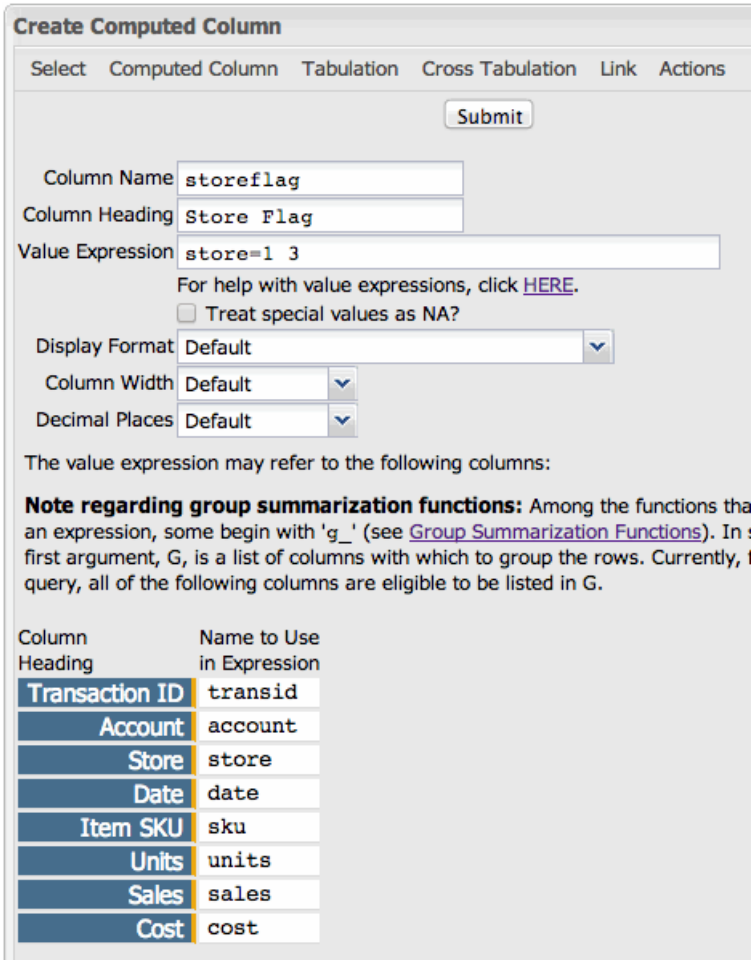
Next, let's look at one more example. This time, we're going to also utilize the `S` parameter of `g_sum(G;S;X)`.

The `S` parameter provides a way for you to filter which rows the G_Function will operate on. Just as we created a selection column to ultimately filter which rows of the summary we viewed in the last example, G_Functions can look at a selection row to know whether or not they should include a specific row in a calculation. As an example, let's say that we want to summarize the sales of stores 1 and 3 in our table, but exclude store 2. This is easy to do with the combination of G_Functions that are given a selection column as a parameter. But in order to do that, first, we have to create the selection column.

Go to **Columns** > **Create Computed Column...** and enter the Value Expression:

```
store=1 3
```

as shown in the screenshot below:



The result will be a column where 1 is the value for rows within the definition of the column (i.e., stores 1 and 3) and 0 for those outside the definition (i.e., store 2).

Sales Item Detail

Columns 1-9 of 9, Rows 1-28 of 35

transid	account	store	date	sku	units	sales	cost	storeflag
531	957	1	05/15/12	366	-1	-5	-1.84	1
532	478	1	05/15/12	98A	1	0.5	0.25	1
532	478	1	05/15/12	3B7	1	1.1	0.56	1
534	738	1	05/16/12	A96	2	6	2.9	1
534	738	1	05/16/12	65B	1	2.25	1.35	1
535	709	2	05/15/12	CB7	1	1.65	1.1	0
535	709	2	05/15/12	96A	1	1.1	1	0
535	709	2	05/15/12	969	1	1.1	1	0
536	748	2	05/17/12	3A4	3	1.02	0.39	0
536	748	2	05/17/12	366	1	5	1.8	0
537	523	3	05/15/12	CB7	1	1.65	1.1	1
537	523	3	05/15/12	A96	2	6	2.9	1
537	523	3	05/15/12	98A	1	0.5	0.25	1
537	523	3	05/15/12	96A	1	1.1	1	1
537	523	3	05/15/12	3B7	1	1.1	0.56	1
538	668	1	05/18/12	CB7	1	1.65	1.1	1
538	668	1	05/18/12	98A	4	2	1	1
538	668	1	05/18/12	96A	1	1.1	1	1
538	668	1	05/18/12	969	1	1.1	1	1
538	668	1	05/18/12	3B7	1	1.1	0.56	1

We can now use the `storeflag` column shown above as the `S` parameter in the `g_sum(G;S;X)` function, as follows:

```
g_sum(store;storeflag;sales)
```

Create a Computed Column and enter the above value expression. Your result will look like this:

Sales Item Detail

Columns 1-10 of 10, Rows 1-28 of 35

transid	account	store	date	sku	units	sales	cost	storeflag	storesales
531	957	1	05/15/12	366	-1	-5	-1.84	1	23.19
532	478	1	05/15/12	98A	1	0.5	0.25	1	23.19
532	478	1	05/15/12	3B7	1	1.1	0.56	1	23.19
534	738	1	05/16/12	A96	2	6	2.9	1	23.19
534	738	1	05/16/12	65B	1	2.25	1.35	1	23.19
535	709	2	05/15/12	CB7	1	1.65	1.1	0	0
535	709	2	05/15/12	96A	1	1.1	1	0	0
535	709	2	05/15/12	969	1	1.1	1	0	0
536	748	2	05/17/12	3A4	3	1.02	0.39	0	0
536	748	2	05/17/12	366	1	5	1.8	0	0
537	523	3	05/15/12	CB7	1	1.65	1.1	1	20.35
537	523	3	05/15/12	A96	2	6	2.9	1	20.35
537	523	3	05/15/12	98A	1	0.5	0.25	1	20.35
537	523	3	05/15/12	96A	1	1.1	1	1	20.35
537	523	3	05/15/12	3B7	1	1.1	0.56	1	20.35
538	668	1	05/18/12	CB7	1	1.65	1.1	1	23.19
538	668	1	05/18/12	98A	4	2	1	1	23.19
538	668	1	05/18/12	96A	1	1.1	1	1	23.19
538	668	1	05/18/12	969	1	1.1	1	1	23.19
538	668	1	05/18/12	3B7	1	1.1	0.56	1	23.19
539	25	2	06/19/12	65B	1	5	3.35	0	0
540	361	2	06/19/12	3B7	1	1.1	0.56	0	0
540	361	2	05/19/12	3A4	1	0.34	0.13	0	0
541	469	3	05/22/12	A96	1	4	1.45	1	20.35
541	469	3	05/22/12	98A	1	0.5	0.25	1	20.35
541	469	3	05/22/12	96A	1	1.1	1	1	20.35
541	469	3	05/22/12	969	1	1.1	1	1	20.35
541	469	3	05/22/12	3B7	3	3.3	1.68	1	20.35

Now we have totals for stores 1 and 3. To narrow down the rows, we can use the `g_first1(G;S;O)` function again, only this time we'll also include our selection column in the `S` parameter:

```
g_first1(store;storeflag;)
```

Here are the results with the new Computed Column:

Sales Item Detail

Columns 1-11 of 11, Rows 1-28 of 35

transid	account	store	date	sku	units	sales	cost	storeflag	storesales	recordflag
531	957	1	05/15/12	366	-1	-5	-1.84	1	23.19	1
532	478	1	05/15/12	98A	1	0.5	0.25	1	23.19	0
532	478	1	05/15/12	3B7	1	1.1	0.56	1	23.19	0
534	738	1	05/16/12	A96	2	6	2.9	1	23.19	0
534	738	1	05/16/12	65B	1	2.25	1.35	1	23.19	0
535	709	2	05/15/12	CB7	1	1.65	1.1	0	0	0
535	709	2	05/15/12	96A	1	1.1	1	0	0	0
535	709	2	05/15/12	969	1	1.1	1	0	0	0
536	748	2	05/17/12	3A4	3	1.02	0.39	0	0	0
536	748	2	05/17/12	366	1	5	1.8	0	0	0
537	523	3	05/15/12	CB7	1	1.65	1.1	1	20.35	1
537	523	3	05/15/12	A96	2	6	2.9	1	20.35	0
537	523	3	05/15/12	98A	1	0.5	0.25	1	20.35	0

Skips store 2 thanks to our friend the selection column

Now we can perform a selection to filter out the rows we don't want:

```
recordflag=1
```

Sales Item Detail

For: recordflag=1

Columns 1-11 of 11, Rows 1-2 of 2

Below: The three columns we created for our analysis.

transid	account	store	date	sku	units	sales	cost	storeflag	storesales	recordflag
531	957	1	05/15/12	366	-1	-5	-1.84	1	23.19	1
537	523	3	05/15/12	CB7	1	1.65	1.1	1	20.35	1

As you can see, G_Functions are specially designed to quickly perform calculations through entire tables. For ease of illustration we used a very small data table where the efficiencies of G_Functions are less important. However, when working on very large data sets you will find that G_Functions are often the best tool at your disposal for data summaries. In the next section we'll delve into the reasons why.

Which Columns to Use for Grouping

After looking at the last few examples, it should be fairly clear that the **G** in G_Functions stands for *Group*.

Whether you're aggregating sales data by store or temperature ranges by city, groups provide the center around which a data summary aggregates. However, for very large data tables not any column can be used as the *Group* argument in a G_Function.

One of the reasons G_Functions are both fast and efficient is that there is an established level of trust between a G_Function and the data table you are working with. By "trust" we simply mean that the G_Function is expecting the data is already arranged in a certain way, and we must make sure that it is before we pass a column to it.

The full explanation of why this is the case would require we delve into a lot of 1010data's system architecture, which is slightly outside the scope of this guide. However, we can address the basic reasons. In order to perform calculations on large datasets, very large tables (over about 4 million rows, give or take) must be broken up, or, *segmented*. This is true of all tables in 1010data that exceed the 4 million row limit. G_Functions must know how a table is segmented in order to operate on the data correctly. So, in order for a column to be usable as the `G` argument in a G_Function, the table must be segmented on that column. This means that all the data for a given group is contained in the same segment. So, for our sales by store example above, if our table was over 4 million rows, we would need to be sure that all records for store 1 are in the same segment. All of store 2 must be in the same segment, and all of store 3 must be in the same segment as well. Stores 1, 2, and 3 can all be in separate segments from one another, but we can't have some records for store 1 in one segment and additional records from the same store in a different segment.

This works because if the G_Function can assume all the records for the `G` parameter are all in the same segment it doesn't need to check anything. It can simply operate across the entire segment and know it hasn't missed any records for the given group.

Your next question should be, "How do I know which column or columns a table is segmented by?" Great question! The good news is the system will tell you exactly which column or columns can be used as your *Group*. Simply go into any of the dialogs we've already used, most notably the **Create Computed Column** dialog, where you will see the following description:

Note regarding group summarization functions: Among the functions that may be included in an expression, some begin with 'g_' (see Group Summarization Functions). In such functions, the first argument, G, is a list of columns with which to group the rows. Currently, for this table and query, the first column(s) in G must be `transid`. The remaining items, if any, may be any of the following columns (other than `transid`).

Column Heading	Name to Use in Expression
Trans ID	`transid`
Date	`date`
Time	`tme`
Store	`store`
SKU	`sku`
Extended Sales	`xsales`
Qty/ Wgt	`qty`
Promo	`promo`
Cost	`cost`
Customer	`customer`
Department	`dept`
Group	`group`
Division	`division`
Sub-Division	`subdivision`
Primary Segment	`primary_segment`
Secondary Segment	`secondary_segment`

G_Function Recap

When you are planning a new data aggregation, summarization, or Quick Query, G_Functions are often the best way to build your analysis. The G_Function library offered by 1010data gives you many more options than standard tabulations for performing calculations across entire data sets. G_Functions are also optimized to work with 1010data's powerful system architecture, so your results are returned faster. And lastly, G_Functions let you retain all the information in your table, as opposed to eliminating columns that didn't apply directly to your summary.

When you do decide to use G_Functions, keep in mind that there are a few rules that will keep you on track. Keep the following things in mind when using a G_Function:

- Make sure you understand how the table you're working with is Segmented
- If you only need to aggregate by specific values for a group, make sure you create a Selection Column
- Know your parameters:
  - `G` is the group you're aggregating by (i.e., store numbers)
  - `X` is the data you are acting on with the G_Function (i.e., adding up sales figures)
  - `S` is the Selection Column that will include or exclude values in a group (i.e., 0 for exclude, 1 for include)
  - `O` is the order in which the results will be sorted
- Use G_Functions in tandem to really create fast, powerful reports and summaries

Of course, we only scratched the surface of G_Functions in this tutorial. Look for future tutorials that will examine how to use more advanced G_Functions, as well as specific analyses that use G_Functions. But for now, start slowly, explore the function library, and build something that answers an interesting question.