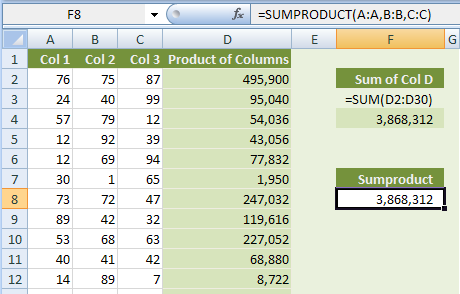
One would never guess from the innocuous name the shear power and versatility embedded within the SUMPRODUCT function. Not only is it well worth your time to examine this function in detail, it is quite literally a prerequisite to honing your Excel abilities to the Excel Hero level. Rarely do I build a model that does not make extensive use of the SUMPRODUCT function.

From a maths perspective, SUMPRODUCT calculates the dot product of two vectors (arrays), but realize that it can be used with many more than two arrays.  A better name might have been the SumOfProducts function, as this in a nutshell is what it does.  Consider the following image:



Notice that Column D is the product of the three columns to its left. Cell D2 contains the formula =A2\*B2\*C2 and that formula is copied down as far as the columns go (Row 30 in this case). The entire purpose for Column D is to get intermediary values so that we can later sum them. Cell F4 shows this final calculation. For some spreadsheets this intermediary column approach is wanted for presentation purposes. However, sometimes all we want is the ultimate total, the sum of all those products. And that's exactly what the SUMPRODUCT function does. Notice the formula in the Formula Bar. It results in the same total.

There are some points to note here.

Firstly, notice that the arrays I've used are the entire columns (A:A, etc.). I've done this on purpose to point out that doing so is a bad idea.  This was not even possible prior to Excel 2007 and would result in an error. Excel 2007 is happy to accommodate the request, but it puts a massive hit on the instance of Excel that workbook is running in. Always try to limit the number of cells that Microsoft Excel functions or formulas must process. A better formula would have been:

=SUMPRODUCT(A2:A30,B2:B30,C2:C30)

The SUMPRODUCT function multiplies elements of different arrays that hold the same position in those arrays and when done, sums all those products. This trivial example made this obvious by showing the function at work on three columns. But understand that those arrays could have just as easily been rows instead of columns, or they could have been rectangular ranges of any arbitrary dimension. They could have been array constants, i.e. {2,3,4,1,9,8} or Named Ranges, or the results of a boolean comparison (simple or complex), a custom mixture of many native Microsoft Excel functions, an Array Function, or even a User Defined Function. The one overriding requirement is that the arrays must be rectangular, contiguous, and of the same dimension. So if Array 1 is a column of 29 elements (as in the above image) then so must be all of the other ranges.

The next point is that this simple example had just three columns (arrays). The SUMPRODUCT function is just as happy with 1 array or up to 30 arrays, each separated by a comma. Thirty is the maximum number of arguments that Microsoft Excel functions can support.

Ok, so SUMPRODUCT is adept at summing the products of equivalently placed elements in arrays. Why is that so powerful? Because it can be used in many non-obvious ways to calculate totals and mine data.

One way that it shines is calculating sums, counts, averages, and other metrics of list data based on criteria. The result here is basically analogous to a SQL database query that produces a total with a WHERE clause. For example the following SQL query:

SELECT SUM(OrderPrice) FROM Orders WHERE Customer='Hansen'

could be emulated in Excel with any one of these SUMPRODUCT formulas:

=SUMPRODUCT((OrderPrice),--(Customer="Hansen"))

=SUMPRODUCT((OrderPrice),1\*(Customer="Hansen"))

=SUMPRODUCT((OrderPrice),(Customer="Hansen")\*1)

=SUMPRODUCT((OrderPrice),(Customer="Hansen")+0)

=SUMPRODUCT((OrderPrice),(Customer="Hansen")^1)

=SUMPRODUCT((OrderPrice),SIGN(Customer="Hansen"))

=SUMPRODUCT((OrderPrice),N(Customer="Hansen"))

=SUMPRODUCT((OrderPrice),(Customer="Hansen")\*True)

=SUMPRODUCT((OrderPrice)\*(Customer="Hansen"))

Some notes are in order here.

The terms OrderPrice and Customer could be Named Ranges (they would basically be defined as columns in a list or table and would each be of the same length). Those Named Ranges (I actually prefer to refer to them as Named Formulas) could be dynamic ranges. Or conversely the terms could be replaced with direct cell references, such as C2:C1000 for OderPrice and B2:B1000 for Customer. Or the terms could be replaced by almost any combination of Microsoft Excel functions that returns an array. Please note that for some reason, SUMPRODUCT always fails when using an IF function on any of the terms inside the SUMPRODUCT, which is OK because we can use boolean logic to eliminate the need for branching. Please read [I Heart IF](http://www.excelhero.com/blog/2010/01/i-heart-if.html) for details.

The order of the terms is irrelevant. In any of the above examples, the answer is the same, whether the OrderPrice term or the Customer term comes first.

The first eight variations above all have two things in common. They use a comma to separate the terms, and each is employing a slightly different technique to coerce a lone boolean value (True or False) into their numeric equivalents, 1 or 0. If we fail to do this coercion on LONE boolean values, the SUMPRODUCT function always returns the value of 0. Of the eight different coercion methods, the double unary method (--) is the quickest.

The ninth variation (the last one) above is the syntax I personally use most often. The entire calculation is done in the first argument so no commas are required and since the boolean term is being multiplied by the OderPrice term, there is no need to further coerce the boolean into a number. The multiplication has already done.

This method is shortest in punctuation and therefore more concise. But it is imperceptibly slower. However, the trade off is greater flexibility. Since all of the work is being done in the first argument to the SUMPRODUCT function, there is no longer a limit of 30 terms. Using this syntax you can combine as many terms as you need. And most important of all, this method allows for more robust logic in the criteria. Specifically, we can specify OR clauses. For example the following SQL query:

SELECT SUM(OrderPrice) FROM Orders WHERE Customer='Hansen' OR Customer='Jones'

could be emulated in Excel with this SUMPRODUCT formula:

=SUMPRODUCT((OrderPrice) \* ((Customer="Hansen")+(Customer="Jones")))

When combining the boolean terms (the ones that evaluate to True or False), they are easy to read if you remember that the plus symbol means OR, while the multiplication symbol means AND. Using this technique, you can tabulate very specific records indeed. For example, consider the following SQL query:

SELECT SUM(OrderPrice) FROM Orders WHERE (Customer='Hansen' OR Customer='Jones') AND State='CA' AND SalesDate>=#1/1/2010# And SalesDate<=#1/31/2010#

This query just says to total all of the orders in January of 2010 from California for the customers Hansen or Jones. It could be achieved in Excel using this SUMPRODUCT formula:

=SUMPRODUCT((OrderPrice) \* ((Customer="Hansen")+(Customer="Jones")) \* (State="CA")\*(SalesDate>="1/1/2010") \* (SalesDate<="1/31/2010"))

For me this is even more "readable" than the SQL database query.

No matter how precise your criteria, which basically means no matter how many criteria terms you need to combine, the SUMPRODUCT function stands ready to tally the specified records, when used with my preferred syntax of combining everything into the first argument.

Another advantage (in my opinion) to using this syntax style is that it acts differently than the comma separated form when there is text in the summing array (i.e. OrderPrice in our examples) in any record returned by the criteria. Sometimes erroneous text can be in a column of data. The comma separated method will treat such text as a 0, combining everything in the first term will result in an error when the text is found in criteria specific records. I prefer to be notified with the error rather than silently calculating an incorrect result.

You'll notice that in all of these examples, we are summing the OrderPrice column. If instead we wanted to count the number of records that satisfied the criteria, all we need to do is drop the OrderPrice term from the formula. So if we wanted to know how many orders there were in January of 2010 from California for the customer Hansen or Jones, we would use this formula:

=SUMPRODUCT(((Customer="Hansen")+(Customer="Jones")) \* (State="CA") \* (SalesDate>="1/1/2010") \* (SalesDate<="1/31/2010"))

Notice that the only difference here is that the OrderPrice term is gone. And finally if we wanted to know the Average amount of those orders we would just divide the total amount by the number of orders. This makes for a long formula, but now that you understand the syntax, it's easy to follow. This very specific average would be calculated with this formula:

=SUMPRODUCT((OrderPrice) \* ((Customer="Hansen")+(Customer="Jones")) \* (State="CA") \* (SalesDate>="1/1/2010") \* (SalesDate<="1/31/2010"))    /    SUMPRODUCT(((Customer="Hansen")+(Customer="Jones")) \* (State="CA") \* (SalesDate>="1/1/2010") \* (SalesDate<="1/31/2010"))

It should be noted here that sometimes when the formulas grow this large, it can be helpful to name them (or parts of them) in the Name Manager.

These type of SUMPRODUCT calcs with multiple criteria can become very interesting when the individual terms are used with OFFSET or INDEX or INDIRECT or MOD/ROW. These enhancements go beyond the scope of this post, but they enable such refinements as considering only every nth record in the tabulations, changing the scope of the array, and many other fascinating feats.

Experienced users may point out that most of the above can be done by using the SUMIF or COUNTIF (or in Excel 2007 SUMIFS and COUNTIFS) functions. This is only partially true. The SUMIF and COUNTIF functions can only work with one criterion. The SUMIFS, COUNTIFS, and AVERAGEIFS function of Excel 2007 can work with up to 29 criteria, but they are limited to AND Logic between the criteria with no way to specify OR Logic.

Really experienced users might point out that all of the above can be done with Array Formulas. While this is true, the SUMPRODUCT function is optimized and is roughly 10% faster than an equivalent Array Formula, and there is no need for the Control-Shift-Enter formula entry.

All of this just scratches the surface of what the SUMPRODUCT function can accomplish. It can be used in countless situations that have nothing to do with database calculations. For example, if you wanted to know if the number in A1 was a prime number, you could use this non-Array Formula:

=SUMPRODUCT(1\*(MOD(A1,ROW(INDIRECT("1:"&INT(A1^0.5))))=0))=1

SUMPRODUCT can be used to do a multi-column sort by formula.

When a SUMPRODUCT function is combined with Conditional Formatting some very interesting results can be achieved.

Another extremely important use of the SUMPRODUCT function which opens an entire universe of new possibilities is to use it to conduct bitwise logical operations such as Logical AND, OR, XOR, NOT, IMP, or EQ. This is the direct manipulation of the bits inside of an integer. There are countless uses of bit manipulation and [this post](http://www.excelhero.com/blog/2010/01/5-and-3-is-1.html) can get you started.

Finally, amongst Excel gurus there is considerable debate about the best way to do database type calculations in Excel. While SUMPRODUCT as demonstrated in this post is very capable, there are alternatives. No one solution is the perfect answer to every situation. I have been conducting timing trials and will soon share detailed results for many different techniques. In the meantime it should be enough to know that SUMPRODUCT is perfectly fine when it is scanning several thousand records, even tens of thousands. However, if you are using it for hundreds of thousands of records than an alternative is likely in order.