Abstract

On-Line Analytic Processing, using Drill Down Techniques has become enormously popular in recent years using GUI front-end programs. Do you know that you have always had the capability to perform standard drill downs using existing tools such as PROC SUMMARY, DATA steps and macros; even without fancy windows and tab buttons? This paper will demonstrate how to develop, create and access a highly summarized database which has been optimized for drill down access. Topics concerning efficiency, flexibility, data integrity, and comparison with other query methods will also be discussed.

Introduction

The modern day usage of “drill down” and OLAP automatically suggests sophisticated Graphical User Interfaces which allow analysts to access various levels of highly summarized data via the click of a mouse. This paradigm is highly desirable, however in many instances it is beneficial to exert more control over the drill down hierarchy. In the SAS® system, it is possible to remove the GUI aspect and control the drill down structure using other means, such as SAS DATA step programming and procedures. In addition, it may be desirable to prototype certain hierarchical models, without the overhead of building or utilizing a GUI. In these and other cases, it is important to understand how to navigate through drill structures by using standard SAS procedures such as PROC SUMMARY.

Definitions

Drill Down

In the context of this paper Drill Down is the process of exploring data by first starting with summarized data and then revealing the detail data contained within that summarized data via a selection process. This detail data may in itself be summarized data, which then can be drilled down to the next level of detail. This process can be continued indefinitely until it reaches a level of detail data which can no longer be summarized. Reversing this process, starting with very detail data, and ending with highly summarized data is referred to as “Drill Up”. The ordering of variables which represent the different levels of detail is referred to as the drill down hierarchy. Each Drill Down action operates upon a specific level of data detail. A desirable aspect of a drill down hierarchy is that the elements contained at each level can be reached via a unique path in the hierarchy. This is not a requirement; however this is done so that the analyst has a clear understanding of the structure of the hierarchy. However, an application can have many drill down hierarchies which represent various dimensions and subject areas.

Numerically we can represent a drill level as a number from 0 to infinity, where 0 represents the most highly summarized level and numbers greater than 0 represent increasingly higher levels of detail. Finally, a well designed drill down application complements the analysts business point of view. This implies that a drill down application cannot be all things to all people and serves a specific business function.

OLAP

OLAP databases allow users to carry out their own analysis, typically across several dimensions, and allow analysts to discover new relationships. This first and most important point of Codd’s 12 OLAP rules places high requirements on software vendors and developers to anticipate and satisfy requirements by analysts (1993). Critics have the opinion that analysts requirements can never be met using some of these products. In any case, drill down applications have been part of the OLAP solution since the early days of the “navigation bar” of the early spreadsheets.

Dimension

A dimension is a category of data. Analysts may wish to look at data across many different dimensions, such as Customer, Product, Date etc. Contrast this with a “Fact” which is a numerical measurement such as salary, weight, sales, etc. One of the purposes of OLAP is the ability to “slice and dice” Facts over many Dimensions.
**Sample Data**

A sample demonstration database is used in the examples which consists of 35,000 observations measuring sales figures collected from various cities in the USA and CANADA. Each observation contains an individual sales amount generated by a specific SALESPERSON for a particular DATE, COUNTRY, REGION, STATE/PROVINCE, and CITY.

**Review of PROC SUMMARY**

PROC SUMMARY is a SAS procedure which can produce various summary statistics for all numeric variables in a data set. This procedure is well suited to OLAP and drill down applications since it can produce output which leads to many different kinds of analyses, both within the same dimension or across different dimensions. In PROC SUMMARY terminology the variables listed on the CLASS statement take on attributes of Dimensions, and variables on the VAR statement are treated as Facts. Data statistics pertaining to Facts can then be pre-summarized, and then be available for analysis with minimum impact upon computation time.

PROC SUMMARY produces separate summary statistics for various subgroups of variables on the CLASS statement. A subgroup is defined for each individual CLASS variable, as well as for each combination of CLASS variables. Thus, 3 variables which are listed on the CLASS statement will generate 8 separate summary subgroups.

The output from PROC SUMMARY is a special data set containing summary statistics which have been calculated for each variable listed on the VAR statement. Important points regarding this output data set are:

- Each observation contains a special _TYPE_ variable which identifies the subgroup to which the summary statistics apply. By examining the _TYPE_ variable as a bitmap variable you can “decode” a specific subgroup and determine which variables on the CLASS statement define that subgroup.
- Each observation contains a _FREQ_ variable representing the number of observations for that current subgroup.
- The observations also contain the CLASS variables which represent the subgroup. If a CLASS variable does not contribute to the subgroup it is represented as missing.

Let’s use the Sample Data Set as an example and set up a PROC SUMMARY with the following code:

```sas
PROC SUMMARY data=drill noprint;
CLASS city state region;
VAR sales;
output out=temp sum=;
```

The output dataset “temp” contains 8 observations which represent sales summaries for all the observations in “drill” broken down by every combination of CITY, STATE, and REGION (Table 1). To see how the _TYPE_ variable works, we will look at the _TYPE_ variable as a bit string with each bit representing a variable in the CLASS statement. If that bit is “on”, the variable it represents is included in the subgroup. The rightmost CLASS variable is represented by the rightmost bit, the next adjacent CLASS variable to the left is represented by the next bit to the left, etc. To arrive at a integer value for _TYPE_, you multiply the value of each bit by 2 raised to the K power, where K represents the bit position, with 0 being the rightmost position and increasing by 1 for each bit to the left. Sum all of the results for every CLASS variable as computed above to arrive at a final value for the _TYPE_ variable.

The 8 different _TYPE_ variables corresponding to the following subgroups are defined below:

<table>
<thead>
<tr>
<th><em>TYPE</em></th>
<th>SUBGROUP</th>
<th>BITSTRING OF CITY STATE REGION</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>REGION</td>
<td>0 0 1</td>
</tr>
<tr>
<td>2</td>
<td>State</td>
<td>0 1 0</td>
</tr>
<tr>
<td>3</td>
<td>Region by State</td>
<td>0 1 1</td>
</tr>
<tr>
<td>4</td>
<td>City</td>
<td>1 0 0</td>
</tr>
<tr>
<td>5</td>
<td>Region by City</td>
<td>1 0 1</td>
</tr>
<tr>
<td>6</td>
<td>State by City</td>
<td>1 1 0</td>
</tr>
<tr>
<td>7</td>
<td>Region by State by City</td>
<td>1 1 1</td>
</tr>
<tr>
<td>0</td>
<td>TOTAL</td>
<td>0 0 0</td>
</tr>
</tbody>
</table>

Example:

Using Table 1 Above: The Subgroup “Region” is computed as:

\[1*2^0 + 0*2^1 + 0*2^2 = 1\]

while Subgroup “State by City” is computed as:

\[0*2^0 + 1*2^1 + 1*2^2 = 6\]

A _TYPE_ variable equal to 0 always represents statistics representing the total for the entire dataset. For each distinct _TYPE_ there will be as many elements in a subgroup as there are levels or combinations of levels for the contributing CLASS variables. For example, if our sample data set contained only two states: NJ, and MD,
and contained only three cities, Baltimore, Newark, and Annapolis, there would be two observations corresponding to \_TYPE\_=2, three observations corresponding to \_TYPE\_=4, and three observations corresponding to \_TYPE\_=6.

**Selection of the Dimensions and Drill Hierarchy**

Selection of the proper dimensions for the drill down involves careful study of analysts needs and an understanding of the relationship among the entities which the data represents. In many cases, the drill hierarchy may follow an organizations standard way of doing business, but in other cases, the data can be mined first using techniques such as CHAID and CART in order to find significant dimensions for analysis. The entire process of determining the needs of the analyst can not be overemphasized, since the drill down model should provide a meaningful application for the analyst. If not, the model can end up being too simplistic, complex, or simply wrong from a business point of view.

Our sample data will use a Time/Geography Drill hierarchy. Examining the relationship among the levels there is a natural relationship as follows:

**drill down**

\[
\text{DATE} \rightarrow \text{COUNTRY} \rightarrow \text{REGION} \rightarrow \text{STATE/PROVINCE} \rightarrow \text{CITY}
\]

**drill up**

\[
\text{CITY} \rightarrow \text{STATE/PROVINCE} \rightarrow \text{REGION} \rightarrow \text{COUNTRY} \rightarrow \text{DATE}
\]

In other more complex cases there may not be a natural order, or the relationship may be a one-to-many relationship. In this case more analysis would be needed.

**Specification of the PROC SUMMARY**

Next, we will set up a PROC SUMMARY specifying the hierarchy we have established. However, we must take care when setting up the CLASS statement, since the order in which the variables appear is important. In order to take advantage of numerical properties of the \_TYPE\_ variable, we will specify the drill down order in the CLASS statement from right to left. Therefore the statement for this example becomes:

\[
\text{PROC SUMMARY data=drill noprint;}
\text{CLASS city state region country date ;}
\]

\[
\text{VAR sales;}
\text{output out=temp sum=;}
\]

Note that we also need to select an analysis variable as well as a summary statistic. Since drill down applications need to access summary statistics at each level of the drill hierarchy, we need to specify \text{SUM=} on the output statement. We will also specify \text{SALES} as the variable which is to be summed.

**Creating the Optimized Drill Down Dataset**

As mentioned earlier, the output of PROC SUMMARY will generated observations for each combination of CLASS variables listed on the CLASS statement. Even though PROC SUMMARY drastically reduces the number of observations in the output dataset, as compared to the original dataset, in a drill down hierarchy we are only interested in a subset of all possible subgroups. We will examine the output dataset “temp” to see which observations correspond to the drill hierarchy at each level. Those observations will be kept, while others will be discarded.

- We need the summary for \text{DATE} by itself (\_TYPE\_=1) since this is the starting level for our drill down.
- The next level of drill down is to the \text{COUNTRY} level. In this case, the required summary would be the one corresponding to \_TYPE\_=3 (DATE by \text{COUNTRY}). We will include \text{DATE} as part of this subgroup since we will already have drilled down from the starting \text{DATE} level, and need the \text{DATE} variable as a way to provide a link to the previous level. If we examine all of the other subgroups which contain \text{COUNTRY} we will find a summary for \text{COUNTRY} by itself, as well as subgroups containing combinations of variables such as \text{COUNTRY} by \text{CITY}, \text{COUNTRY} by \text{STATE} by \text{CITY}, etc. Some of these summaries will be discarded, since we are only interested in that summary relevant to positions within the drill down hierarchy.
- If we continue to work through the drill hierarchy to the next level, we see that the \text{REGION} level requires \_TYPE\_=7 (DATE by \text{COUNTRY} by \text{REGION}), since by drilling down to the \text{REGION} level, we will have already drilled down on \text{DATE} and then \text{COUNTRY}. By the same rational given for the previous level, we can discard the other \_TYPE\_ variables containing \text{REGION} not relevant to our drill down. (For example: \text{REGION} by \text{CITY} will be
discarded since REGION is never selected by drilling down on CITY).

- If we continue this process to the last variable, CITY, we see that the only _TYPE_ values needed are 1, 3, 7, 15 and 31. At each of these levels, the subgroup includes the CLASS variables corresponding to the variable at that level, plus all variables which have already been “drilled down on” in the hierarchy.

If we only include these types, how much will that reduce the size of the output dataset? That will depend upon the number of distinct levels for each _TYPE_. Referring to Table 2, we can see that we are now left with a dataset containing only 68 observations, drastically reducing the number of observations in the output summary data set, relative to both the original “drill” dataset, and the output “temp” dataset.

Table 2

<table>
<thead>
<tr>
<th><em>TYPE</em></th>
<th>SUBGROUP</th>
<th>OBS</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Date</td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td>Country by Date</td>
<td>10</td>
</tr>
<tr>
<td>7</td>
<td>Region by Country by Date</td>
<td>15</td>
</tr>
<tr>
<td>15</td>
<td>State by Region by Country by Date</td>
<td>34</td>
</tr>
<tr>
<td>31</td>
<td>City by State by Region by Country by Date</td>
<td>34</td>
</tr>
</tbody>
</table>

We can generalize the above example by stating that in a drill down hierarchy, where we have \( N \) CLASS variables which have been arranged in drill down order from right to left, in order to perform a drill down as described above, we only require the output observations which have _TYPE_ values in:

\[
\{ 2^0, \ 2^0 + 2^1, \ 2^0 + 2^1 + 2^2, \ 2^0 + 2^1 + 2^2 + 2^3 \ldots \ldots \ldots, \ 2^0 + 2^1 + 2^2 + 2^3 + 2^{N+1} \} \]

with 0 being the most rightmost position, and increasing by 1 for each variable to the left. \( (1.1) \)

Using the output data temp as generated above we create the optimized drill down dataset by subsetting:

```sql
data temp;
set temp;
where _TYPE_ in (1, 3, 7, 15, 31);
```

A special form of the above formula exists when the drill hierarchy starts from the rightmost variable in the CLASS statement. In this case the first _TYPE_ value is always 1, and the next drill down value is generated by multiplying the last variable by 2 and adding 1. \( (1.2) \)

In a “drill-up” situation, this would be equivalent to subtracting 1 from the current level and then dividing by 2.

If we still are left with a large output data set, after reducing the size in this manner, we may choose to place an index on the _TYPE_ variable if it will provide significantly quicker access to the data.

```sql
PROC SQL;
create index _TYPE_ on temp (_TYPE_);
```

Reordering the Drill Hierarchy

It is usually best to set up the order of the drill variables from right to left, as we just did in the preceding example, however it is not essential to do so.

Let’s say we were interested in reordering the drill hierarchy starting with COUNTRY, and then drilling down on DATE, REGION, all the way down to CITY. Our drill hierarchy now becomes:

COUNTRY → DATE → REGION → STATE → CITY

Since we have already created the optimized database, how does that affect the reordering of the variables? We could always reorder the positions of the drill variables from right to left, and rerun the original PROC SUMMARY etc., but there is another way:

If we express our modified drill hierarchy with respect to powers of 2 we have:

\[2^1 \rightarrow 2^0 \rightarrow 2^3 \rightarrow 2^4\]

If we examine the exponent of 2 in each case, we see that it represents the position, starting with 0, of the level in the CLASS statement.

Since COUNTRY is now the 2nd variable from the right on the CLASS statement, it has a _TYPE_ value of 2^1, or 2. This becomes the first _TYPE_ value. The next variable in the hierarchy is DATE, which has a _TYPE_ value of 1, so we add these two values together to get the next _TYPE_ value which is 3. The next level is REGION, so we add the _TYPE_ value of 4 to get the
result 7. Since the remainder of the hierarchy remains the same, the set of new _TYPE_ values generated with respect to the Drill hierarchy now becomes:

\[ \{2, 3, 7, 15, 31\} \]

Note that now we are applying an additive rule which adds the value of the next _TYPE_ value to the summation of the previously accumulated _TYPE_ values. (1.3)

If we look at the set of \( \{2, 3, 7, 15, 31\} \), it does break our prior rules for generating the _TYPE_ values \( \{1.1, 1.2\} \). That is because the drill hierarchy is not being interpreted strictly from right to left.

If we had expressed our original example in the format:

\[ 2^0 \rightarrow 2^1 \rightarrow 2^2 \rightarrow 2^3 \rightarrow 2^4 \]

we could have also used this additive rule to arrive at the identical series \( \{1, 3, 7, 15, 31\} \).

**Analysis of the optimized data set**

Now that an optimized drill data set has been created, we can demonstrate its usage by using simple PROC PRINT statements with WHERE clauses. Of course, this can be extended to other SAS procedures as well. In order to contrast this method with the standard GUI interpretation of Drill down, we will contrast each step of the process with the equivalent GUI application step. For this example we will use the sample data set described above.

**Drilling Down - The Traditional GUI Way**

1. We open up an GUI application which contains sales data for the sample data set described.
2. A list box which contains months and associated sales data for that month is displayed (Table 3).
3. Select May 97 from the list of available months and drill down to the next level which displays a list box containing sales data for the USA and CANADA (Table 4).
4. Select USA from the list of countries to drill down to the next level displaying the sales data for the 2 regions NE, SE (Table 5).
5. Select NE from the list of regions to display the sales data for the 3 states in the Northeast CT, NJ, and NY (Table 6).
6. Select NJ to display the sales figures for Trenton, the only city in the territory with sales data for May 97 (Table 7).
7. We realize that we neglected to note the sales totals for the states in the Northeast, so we click the “BACK” button to get to the previous screen, and note the totals for the states (Table 6).

**Drilling Down - The Windowless Way**

1. There is no front end application to open up. We open up an interactive SAS session, and assign the correct LIBNAMES for the datasets.
2. To obtain a list of sales months and sales generated, corresponding to step 2 above, we first need to determine which level we will start at. In this case it is the DATE level, and we already have established that this corresponds to _TYPE_=1:

   ```
   proc print data=temp;
   var date sales;
   where _TYPE_=1;
   ```

   Switching to the output window reveals all of the months and sales figures which correspond to the results illustrated in Table 3 above. Although the display methods are completely different in both cases, the information is identical.

3. In the previous illustration we selected 01MAY97 from the list of available with a mouse. Let’s simulate the selection of a month in our “windowless” environment.

   ```
   proc print data=temp;
   var country sales;
   where _TYPE_=1;
   ```

   Proc print data=temp;
   var country sales;
   where month='01MAY97'D and _TYPE_=3;
The month which is being selected is supplied via the WHERE statement.
The next drill down level is also supplied in the WHERE statement via the _TYPE_ variable. But in order to determine the value of that _TYPE_ variable, we first need to compute the value of the next drill down level. From our earlier algorithm (1.2), we know that if we are at level 1, the next drill down level will be \((1 \times 2) + 1\) or 3. Therefore, we add the “and _TYPE_=3” clause to the WHERE statement.

We have just performed our first “windowless” drill down, and the output from the PROC PRINT corresponds with the results shown in Table 4.

To generalize the above two steps, in order to simulate the selection and display of the next drill down level you need to:

I. Supply a selection value for the variables associated with the current level \(N\). This is the variable you would “click on” in a GUI application.

II. Determine the value of the _TYPE_ variable for the next drill down level.

III. Subset the data with the above conditions using “WHERE” clauses or “IF” statements, and process the data.

4. Continuing down the drill hierarchy we will select USA from the list of countries. Since the last value of the _TYPE_ variable was 3, the next value will be \((3 \times 2) + 1\) or 7. The output corresponds to Table 5.

Proc print data=temp;
var region sales;
where month='01MAY97'D
and country='USA' and _TYPE_=7;

5. Select NE from the list of regions to drill down to the next level. Results are the same as in Table 6.

Proc print data=temp;
var state sales;
where month='01MAY97'D and
country='USA' and region='NE' and _TYPE_=15;

6. To get to the final detail level we select NJ from the list of cities to obtain the same results as in Table 7.

Proc print data=temp;
var city sales;

7. To simulate the “BACK” button, i.e. to drill up, we compute the new _TYPE_ value as \((31 - 1) / 2\) or 15. The results are identical to the results shown in Table 6.

8. 

Drill Down Data Integrity

In the previous example, we supplied values for all levels that we had already selected when drilling down to the next level. This was done to enforce data integrity dealing with possible “many to one” relationships, and is always recommended. In the sample data, each city is always be contained in the same state, and that every state is always contained in the same region, etc. In these cases it is not necessary to supply all of the values for the previous levels of the drill hierarchy. The condition “where month='01MAY97'D and _TYPE_=31” would be sufficient to show the same results as the one specified in step 6. That is because adding country='USA' and region='NE' is redundant in this case. However, there are many examples which would yield erroneous results if specified this way, and specifying all values for the level and previous levels would be required. Dimensions involving time are an example. If one would drill down from YEAR → MONTH → DAY, one would always have to supply the YEAR, MONTH, and DAY in its entirety when satisfying the WHERE clause. That is because any particular day can occur in many months, month in years, etc. Also, cities with duplicate names, such as Wilmington DE, or Wilmington NC, are examples in which it would be necessary to supply the state name as part of the query.

Crossing Dimensions

In OLAP applications it is very desirable to be able to switch the dimensional view at any time in the analysis. In “windowless” OLAP that is also very possible, given a well designed database, and a knowledge of the drill down hierarchy. Let’s add a 3rd dimension to our sample data, “Product” which contains 2 variables: Product, and Product Size. At any time during drill down of the Time/Geography dimension, the analyst might wish to see
a breakdown by Product, or Product Size. To illustrate how this would work in our context we first need to add the new dimensions to the database. We will follow the same steps for creating the optimized database in this example, as we did in the previous examples.

**Selection of the Dimensions and Drill Hierarchy (2)**

We will define two product variables. The Analyst will be presented with a list of Products (WIDGET1, WIDGET2) and associated sales revenue, from which one product is selected. This displays another list of different sizes for that product (Small, Large), with associated sales figures. The drill hierarchy is defined very simply as:

PRODUCT → SIZE

**Specification of the PROC SUMMARY (2)**

We need to append the two variables PRODUCT and SIZE, from right to left as we did before to the original PROC SUMMARY statement, along with the original variables:

```
PROC SUMMARY data=drill noprint;
CLASS SIZE PRODUCT city state region country date ;
VAR sales;
output out=temp sum=;
```

**Creating the Optimized Drill Down Dataset (2)**

Again, we are only concerned with the specific drill hierarchy, so that in addition to _TYPE_ 1,3,7,15,31 created in the last example, we also need to add the PRODUCT dimensions to the optimized database. As in the previous examples, we are interested only in a subset of the observations output from PROC SUMMARY:

- The _TYPE_ rows generated for the dimension Geography {1,3,7,15,31}
- The _TYPE_ rows generated for the dimension Product by itself {32,96}
- The _TYPE_ rows generated for the cross dimensions Geography * Product

When defining the values for the PRODUCT dimension, the values of the _TYPE_ variables is computed as $2^3$ and $(2^5 + 2^6)$, since 5 and 6 correspond to the position of the variables in the CLASS statement, starting with 0.

When defining the _TYPE_ observation for the Geography * Product dimensions, you can apply an additive rule, by adding the value of the _TYPE_ of the 1st dimension to the value of the 2nd dimension to get the value of the “crossed” dimension. This allows us to switch to PRODUCT view, from any GEOGRAPHY view by adding the value of 32 to the current value of the _TYPE_ variable. These additional PRODUCT/SIZE/GEOGRAPHY types are shown below.

<table>
<thead>
<tr>
<th><em>TYPE</em></th>
<th>SUBGROUP</th>
</tr>
</thead>
<tbody>
<tr>
<td>32</td>
<td>PRODUCT</td>
</tr>
<tr>
<td>96</td>
<td>Product by Product Size</td>
</tr>
<tr>
<td>33</td>
<td>Product by Date</td>
</tr>
<tr>
<td>35</td>
<td>Product by Country</td>
</tr>
<tr>
<td>39</td>
<td>Product by Region</td>
</tr>
<tr>
<td>47</td>
<td>Product by State</td>
</tr>
<tr>
<td>63</td>
<td>Product by City</td>
</tr>
<tr>
<td>97</td>
<td>Product by Product Size by Date</td>
</tr>
<tr>
<td>99</td>
<td>Product by Product Size by Country</td>
</tr>
<tr>
<td>103</td>
<td>Product by Product Size by Region</td>
</tr>
<tr>
<td>111</td>
<td>Product by Product Size by State</td>
</tr>
<tr>
<td>127</td>
<td>Product by Product Size by City</td>
</tr>
</tbody>
</table>

Therefore, to create this optimized data set we will take the output from PROC SUMMARY and apply a DATA step:

```
data temp;
set temp;
where _TYPE_ in (1,3,7,15,31,32,96,33,35,39,47,63,97,99,103,111,127);
```

**Analysis of the optimized data set (2)**

**Examples**

1) Refer to Table 6 where we displayed sales figures for May 1997, for each state in the NE region of the USA. Looking at the output, we decide that we know wish to add product information to the each state sales figures. Since we are already at a level with _TYPE_ =15, and wish to add product to the display (_TYPE_ =32) , we compute the next _TYPE_ = 15 + 32 or 47, and modify the WHERE statement:

```
Proc Print data=temp;
var state product sales;
where month='01MAY97'D and country='USA' and region='NE' and _TYPE_ = 47;
```
<table>
<thead>
<tr>
<th>STATE</th>
<th>PRODUCT</th>
<th>SALES</th>
</tr>
</thead>
<tbody>
<tr>
<td>CT</td>
<td>WIDGET1</td>
<td>2,000,000</td>
</tr>
<tr>
<td>CT</td>
<td>WIDGET2</td>
<td>4,000,000</td>
</tr>
<tr>
<td>NJ</td>
<td>WIDGET1</td>
<td>250,000</td>
</tr>
<tr>
<td>NJ</td>
<td>WIDGET2</td>
<td>750,000</td>
</tr>
<tr>
<td>NY</td>
<td>WIDGET1</td>
<td>2,000,000</td>
</tr>
<tr>
<td>NY</td>
<td>WIDGET2</td>
<td>3,000,000</td>
</tr>
</tbody>
</table>

2) Referring to Table 6, had the analyst wanted to drill down on NJ to the city level, as previously illustrated, and also simultaneously add PRODUCT and SIZE to the mix, our code would become:

```
Proc print data=temp;
  var city product size sales;
  where month='01MAY97'D and country='USA' and region='NE' and STATE='NJ' and _TYPE_=127;
```

<table>
<thead>
<tr>
<th>CITY</th>
<th>PRODUCT</th>
<th>SIZE</th>
<th>SALES</th>
</tr>
</thead>
<tbody>
<tr>
<td>Trenton</td>
<td>WIDGET1</td>
<td>S</td>
<td>300</td>
</tr>
<tr>
<td>Trenton</td>
<td>WIDGET1</td>
<td>L</td>
<td>400</td>
</tr>
<tr>
<td>Trenton</td>
<td>WIDGET2</td>
<td>S</td>
<td>100</td>
</tr>
<tr>
<td>Trenton</td>
<td>WIDGET2</td>
<td>L</td>
<td>200</td>
</tr>
</tbody>
</table>

3) In this example, start the drill down exclusively in the PRODUCT dimension:

```
Proc print data=temp;
  var product sales;
  where _TYPE_=32;
```

<table>
<thead>
<tr>
<th>WIDGET1</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>90,000,000</td>
</tr>
<tr>
<td>WIDGET2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>81,300,000</td>
</tr>
</tbody>
</table>

Now we wish to cross over to the TIME/GEOGRAPHY dimension, displaying Product Sales by Month. In this case we add the _TYPE_ value for DATE (1) to the _TYPE_ value for PRODUCT(32).

```
Proc print data=temp;
  var product date sales;
  where _TYPE_=33;
```

<table>
<thead>
<tr>
<th>WIDGET1</th>
<th>JAN 97</th>
<th>15,500,000</th>
</tr>
</thead>
<tbody>
<tr>
<td>WIDGET1</td>
<td>Feb 97</td>
<td>19,000,000</td>
</tr>
<tr>
<td>WIDGET1</td>
<td>Mar 97</td>
<td>20,000,000</td>
</tr>
<tr>
<td>WIDGET1</td>
<td>Apr 97</td>
<td>18,500,000</td>
</tr>
<tr>
<td>WIDGET1</td>
<td>May 97</td>
<td>17,000,000</td>
</tr>
<tr>
<td>WIDGET2</td>
<td>Jan 97</td>
<td>17,500,000</td>
</tr>
<tr>
<td>WIDGET2</td>
<td>Feb 97</td>
<td>17,000,000</td>
</tr>
<tr>
<td>WIDGET2</td>
<td>Mar 97</td>
<td>16,000,000</td>
</tr>
<tr>
<td>WIDGET2</td>
<td>Apr 97</td>
<td>15,500,000</td>
</tr>
<tr>
<td>WIDGET2</td>
<td>May 97</td>
<td>15,300,000</td>
</tr>
</tbody>
</table>

In all of the above examples we have complete flexibility as to whether we choose to stay in one dimension, cross over to another dimension and traverse its drill hierarchy, or drill down through a particular dimension while simultaneously adding the attributes of another dimension.

### Application Extensions

#### Addition of other statistics to the output data set

Aside from summarization, PROC SUMMARY allows computation of other useful statistics, which can be very helpful in implementing OLAP without windows. Analysts often look for outliers in the data, or look for observations which are greater than some mean value. In these kinds of cases, other additional statistics may be added to the output data set. Of particular importance are the MIN, and MAX statistics. If we were to redefine our original PROC SUMMARY to do this, our code would be:

```
PROC SUMMARY data=drill noprint;
  CLASS city state region country date ;
  VAR sales;
  output out=temp sum=sumsales mean=musales min=minsales max=maxsales;
```

### Macro Applications

Once the optimized data set has been constructed, you can implement techniques to automate the drill techniques as described above. One way to do this is through user defined macro functions such as drillup(), drilldown(), getlevel(), etc. Even though the construction of these macros is beyond the scope of this paper, at minimum the macros would need to compute the proper drill levels. The drill hierarchy could be interpreted by parsing the CLASS statement, or via the order of the _TYPE_ values themselves, and could be stored as a series of macro variables. In conjunction with the algorithms discussed above, a metabase could be constructed to implement intelligent macro calls.

### SAS/AF Integration

I have already touched on the advantages that these techniques offer in terms of implementing a drill down structure for an application before the front-end GUI has been developed. Additionally, once the optimized drill
down database has been built, it is possible to integrate and access the database directly from within a SAS/AF Frame application, possibly using the customized macro calls mentioned above, and Screen Control Language. Features common to such applications might include:

- Identifying the drill down level. This could be determined via PMENU processing or by using SCL functions to determine which drill elements have been selected from within List Boxes, etc.
- Once the drill level has been determined, the appropriate _TYPE_ value could then be computed, through the use of customized macros, or SCL lists.
- The analyst will just about always want to subset the data. A sub-application could be built which constructs a Where Statement based upon user input and the parameters computed above, and submits the entire request to the optimized drill down database using either SUBMIT CONTINUE, or SUBMIT SQL processing.

Whatever method is used to implement integration with SAS/AF Frame applications, development could benefit from cost savings and increased productivity, since the drill down access techniques and methodology will have already been implemented and tested.

**Comparison with other methods**

For comparison purposes, I have identified three alternates for doing drill downs compared to the methods I have outlined. Each has distinct advantages and disadvantages:

**Comparison with PROC SQL**

It is also possible to use Structured Query Language as an interface to drill down hierarchies via the GROUP BY clause. In fact, SQL is the standard query interface for many vendors’ OLAP applications. PROC SQL would need a mechanism for determining the drill down hierarchy, but once determined, could easily be expressed due to the elegance of SQL. Since speed of query would probably be an issue for larger datasets, when using this method, first summarize the data to the highest level of summarization that yields the lowest level of detail for the drill hierarchy. Using the optimized drill down database has an advantage since data has already been presummarized and optimized for Drill Down access.

**Compared with other EIS applications**

You may decide to set up a metabase, drill down structure, and report gallery using SAS/EIS, or to construct multidimensional cubes using one of the many specialized OLAP tools and products in the market today. However these products are not available on all users’ desktop, and do require significant effort to get them to run the way analysts require. How many times have you heard “Well, this is great, but for this specific group (or month, or sample etc.) I need to look at the data in a slightly different manner” ? Contrast that with the widespread availability of BASE SAS, the open structure of the methods, and the equivalent amount of work which would be used to build the optimized drill down database customized to the analysts requirements.

**Brute Force**

Finally, we have the brute force method. Applying each drill down level to the original detail database. Advantages to this method are those dealing with data currency, data integrity, and the need to satisfy many different kinds of requests. The disadvantages are numerous. Many times there is no well thought out plan concerning the levels of the data, and as a result queries can take a long time to run. Multiple programmers can come up with different results due to the complexity of the detail data. Contrast this with the optimized drill down database method which imposes a logical order on the data, and can return results extremely quickly.

**CONCLUSION**

Every situation demands an optimal solution, and drill down applications are no exception. When deciding to whether or not to implement techniques such as the ones described, evaluate your application to determine the advantages and disadvantages of using the optimized drill down database.

**Advantages**

- Speed - since the information has already been highly summarized and data reduction has been performed, queries will run against the data extremely fast. Processing time will usually not be an issue, and the analyst can spend more time analyzing, refining the analysis, and presenting the data.
- Size - Space required to store the data will be minimal. Since the data is restricted to the levels
defined in the drill down hierarchy, the space required to store the data is even less that if the raw output from PROC SUMMARY is used.

- Flexibility - Since the summarized data is derived from the detail data, any additional variables can quickly be added to the summary data set without significant impact of speed, or size. Additionally, there is no need to go through multiple levels of drill down, if you directly specify which level you want to process. Multiple Drill downs hierarchies can be defined, with defined cross-dimension capabilities.

- New drill hierarchies can be prototyped using this method before they are introduced into applications. Summary data sets created by this method can be queried directly by AF applications, saving the need to develop separate data structures.

- Ability to easily combine dimensions of different types into one drill hierarchy. No need to establish separate dimension tables.

**Disadvantages**

- Need to run PROC SUMMARY every time the underlying data changes.

- Knowledge of the value of the _TYPE_ value is required at all time. This can be overcome by using macro techniques and assigning symbolic names.

- The optimized summary dataset is only meaningful with respect to the drill down hierarchy. If the drill down hierarchy is not used, or is restrictive in any way, it does not make sense to use this technique.

**References**


SAS, SAS/AF, SAS/EIS are registered trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

Other brand and product names are registered trademarks or trademarks of their respective companies.

---

**Author Contact**

Ralph Winters
Alydon Technical Services, Inc.
20 Essex Rd, Suite 2A
Maplewood, NJ 07040
973-275-6248
ralph_winters@internetmci.com