Paper A02-2009

Using PROC OLAP to Build Cubes with NON-Additive Measures

Ben Cochran, The Bedford Group, Raleigh, NC

ABSTRACT

Most of the time, OLAP cubes are built from data that has additive measures, meaning that as you drilldown, the sum of all the lower levels will add up to the value at the highest level of the hierarchy. This is not always the case. Sometimes applications need drilldown capabilities on data where the measures are non-additive. And, sometimes data is additive in one dimension, but not another. Take for example, a car leasing company that has 2,000 cars to lease. They want to build a cube with two dimensions: **Time** and **Geography**. Across the Geography dimension, the number of cars is additive. Let's say that the levels in the Geography dimension are: Company, Region, State and City. At the Company level, the number of cars is 2,000. When we drilldown to the Region level, the total at all the regions adds up to 2,000. When we drill down to the next level (State), the total number of cars in all the states adds up to 2,000. etc. This same measure (total number of cars) is NOT additive in the Time dimension. Let's say that the levels of the Time dimension are: Year, Quarter and Month. If we take the number of cars that are leased each Month, they could add up to more than 2,000. And likewise, if we add up all the cars leased each Quarter, they could add up to more than 2,000. But, still this company wants to build a cube with this data. This paper looks at strategies and methods to building a cube with non-additive data. Then, a step by step approach is taken to actually build the cube.

INTRODUCTION

Scenario: a Car leasing company has 2000 cars to lease which are distributed amount 36 cities (in 9 states within 3 regions). They want to build a cube that will tell them how many cars are leased and how many are available as they drilldown through time and geography. There are four measures that they want to follow: Available_Cars, Leased_Cars, Amount_Billed, and Amount_Collected. Available_Cars and Leased_Cars are additive in the Geographic hierarchy, but not in the Time hierarchy. For example, you can add up all the available cars in all the states (geographic hierarchy) and that will give you 2,000. But, no matter what year you are examining (time hierarchy), there are only 2,000 cars available in any given year AND only 2,000 cars available for all the years. (You can't add up that number across the years. Amount_Billed and Amount_Collected are additive across both the Time and the Geography dimension.

THE DATA

The data is stored in a SAS dataset named SASUSER.CUBE_DATA.

VIEWTABLE: Sasuser.Cube_data

	year	month	Region	Stat	city	Amount_Billed	Amount_Collected	Available_Cars	Leased_Cars
1	2008	1	East	NY	Albany	3500	2171	54	35
2	2008	2	East	NY	Albany	3500	3170	55	35
3	2008	3	East	NY	Albany	4300	4000	56	43
4	2008	4	East	NY	Albany	4300	3483	57	43
5	2008	5	East	NY	Albany	5300	4293	58	53
6	2008	6	East	NY	Albany	5300	3360	59	53
7	2008	7	East	NY	Albany	5300	4293	60	53
8	2008	8	East	NY	Albany	5300	4293	61	53
9	2008	9	East	NY	Albany	5300	4293	62	53
10	2008	10	East	NY	Albany	6200	4993	63	62
11	2008	11	East	NY	Albany	6200	5022	64	62
12	2008	12	East	NY	Albany	6200	5022	65	62
13	2009	1	East	NY	Albany	5300	4293	54	53
14	2009	2	East	NY	Albany	5400	4374	55	54
15	2009	3	East	NY	Albany	5400	5100	56	54

Figure 1. Source Data

There is one row per city / month. There are 36 cities with 24 months worth of data for each city for a total 864 rows.

DATA PREPARATION

The above data needs to be manipulated so that we can create two dimensions for the cube: Time and Geography. For the Time dimension, the levels are YEAR and MONTH and for the Geography dimension, the levels are REGION, STATE and CITY. The right four columns (AMOUNT_BILLED, AMOUNT_COLLECTED, AVAILABLE_CARS, and LEASED_CARS) are all measures.   Since AVAILABLE_CARS and LEASED_CARS are NON-ADDITIVE across the TIME dimension, we need to summarize the data and create a SAS dataset for each level of the dimensions.

To do the necessary summarization, both PROC MEANS and the DATA step are used.

```
proc means data=in_cube.cube_data noprint sum chartype;
   var Amount_Billed Amount_Collected Available_Cars Leased_Cars;
   class Year Month Region State City;
   output out=in_cube.YM_11000(where=(_type_='11000') drop= _F:)
          sum(Amount_Billed)=    sum(Amount_Collected)=
          sum(Available_Cars)=   sum(Leased_Cars)=;
   output out=in_cube.YMR_11100(where=(_type_='11100') drop= _F:)
          sum(Amount_Billed)=    sum(Amount_Collected)=
          sum(Available_Cars)=   sum(Leased_Cars)=;
   output out=in_cube.YMRS_11110(where=(_type_='11110') drop= _F:)
          sum(Amount_Billed)=    sum(Amount_Collected)=
          sum(Available_Cars)=   sum(Leased_Cars)=;
   output out=in_cube.YMRSC_11111(where=(_type_='11111') drop= _F:)
          sum(Amount_Billed)=    sum(Amount_Collected)=
          max(Available_Cars)=   max(Leased_Cars)=;
run;
```

Program 1. PROC MEANS

This one PROC MEANS step creates four different SAS datasets all at different levels of summarization. Notice the naming convention of the datasets.   The first dataset to be created is called YM_11000 and is summarized at the YEAR and MONTH level. The key to the numeric pattern is the order of the variables on the CLASS statement.

LEVEL 1 Data

The first 13 rows of YM_11000 are shown below.

VIEWTABLE: In_cube.Ym_11000

	year	month	Region	State	city	_TYPE_	Amount_Billed	Amount_Collected	Available_Cars	Leased_Cars
1	2008	1				11000	132000	110209	1604	1320
2	2008	2				11000	135200	109404	1640	1352
3	2008	3				11000	141300	110503	1676	1413
4	2008	4				11000	145400	119274	1712	1454
5	2008	5				11000	151000	119654	1748	1510
6	2008	6				11000	152600	119923	1784	1526
7	2008	7				11000	156100	125085	1820	1561
8	2008	8				11000	159300	124121	1856	1593
9	2008	9				11000	161900	130160	1892	1619
10	2008	10				11000	170300	142677	1928	1703
11	2008	11				11000	173100	136621	1964	1731
12	2008	12				11000	176200	139863	2000	1762
13	2009	1				11000	151000	121773	1604	1510

Figure 2 : Level 1 Data

Notice the values for the **twelfth** observation. This row represents the total Amount_Billed and total Amount_Collected for ALL of 2008. The other columns (Available_Cars and Leased_Cars) which are NON-Additive, reveal the values at the END of the 2008. In other words, in 2008, we had 2000 cars to lease, and we leased 1,762 of them. So, this is the row we want to show when we are looking at data for the year of 2008.
In order to get these values when we drilldown, we need to run the following code.

```
/* Level 1 : 10000  */
data in_cube.Y_10000  in_cube.All_00000;
    drop AB AC AA LC YAB YAC YAA YLC;
    set in_cube.ym_11000(rename=(Amount_Billed=AB   Amount_Collected=AC
                           Available_Cars=AA  Leased_Cars=LC)) end=e;
    by year;
    if first.year then do;
       Amount_Billed =0;  Amount_Collected=0;
       Available_Cars=0;  Leased_Cars=0;
    end;
    Amount_Billed + AB;  Amount_Collected+AC;
    Available_Cars + AA; Leased_Cars + LC;
    YAB + AB;  YAC + AC; YAA + AA;  YLC + LC;
    if last.year then do;
       _TYPE_ = '10000';
       Month=. ;
       Leased_Cars=LC;
       Available_Cars=AA;
       output in_cube.Y_10000;
    end;
    if e;
    Year=.;  Month=.;  _TYPE_='00000';
    Amount_Billed = YAB;  Amount_Collected = YAC;
    Available_Cars = AA;  Leased_Cars = LC;
    output in_cube.All_00000;
  run;
```

Program 2.

Examine the two datasets created above.

VIEWTABLE: In_cube.All_00000

	year	month	Region	State	city	_TYPE_	Amount_Billed	Amount_Collected	Available_Cars	Leased_Cars
1	.	.				00000	3841500	3073797	2000	1820

Figure 3: ALL_00000 Dataset

The **ALL_00000** data set contains maximum values for all four measures. In other words, the total Amount_Billed for ALL transactions was $3,841,500. The total Amount_Collected for ALL transactions was $3,073,797. The total number of Available_Cars (for both years) was 2000, and the total number of Leased_Cars (for both years) was 1,820.

VIEWTABLE: In_cube.Y_10000

	year	month	Region	State	city	_TYPE_	Amount_Billed	Amount_Collected	Available_Cars	Leased_Cars
1	2008	.				10000	1854400	1487494	2000	1762
2	2009	.				10000	1987100	1586303	2000	1820

Figure 4: Y_10000 Dataset

The **Y_10000** data set is summed for each value of YEAR. Here it may be a little more obvious that Available_Cars and Leased_Cars are NON-Additive. There are only 2,000 cars to lease for BOTH years, not for EACH year.

For classification purposes, the above datasets are referred to as LEVEL 1 datasets.

LEVEL 2 Data

The following code creates the LEVEL 2 datasets. These are summarized for each YEAR and REGION.

```
/*  Level 2:  10100  */

proc sort data=in_cube.ymr_11100;
     by year region month;
run;

data test2m;
   set ymr_11100;
   by year region ;
   if last.region;
run;

proc means data=ymr_11100 sum nway noprint;
     class year region;
     var Available_Cars Leased_Cars Amount_Billed Amount_Collected;
     output out=test2y sum=;
run;

data in_cube.Y_R_10100(drop=_Fre: ) ;
     merge test2y(drop=Available_Cars  Leased_Cars  _type_)
           test2m(drop=Amount_Billed   Amount_Collected _type_);
     by year region;
     month=.;
     _Type_='10100';
run;
```

Program 3.

The first DATA step gets the last row for each REGION for each YEAR. The PROC MEANS step gets the SUM for each REGION for each YEAR. The last DATA step merges the two datasets together so that each row has the SUMS for AMOUNT_BILLED and AMOUNT_COLLECTED and the LAST ROW for AVAILABLE_CARS and LEASED_CARS.   The resulting dataset is shown below.

.

VIEWTABLE: In_cube.Y_r_10100

	year	Region	Amount_Billed	Amount_Collected	month	State	city	Available_Cars	Leased_Cars	_Type_
1	2008	Central	531100	422478	.			581	497	10100
2	2008	East	733500	588021	.			829	715	10100
3	2008	West	589800	476995	.			590	550	10100
4	2009	Central	572400	452909	.			581	531	10100
5	2009	East	817700	661473	.			829	732	10100
6	2009	West	597000	471921	.			590	557	10100

Figure 5: Y_R_10100 Dataset.

The YMR_11100 created by the first PROC MEANS dataset is also a LEVEL 2 dataset.

LEVEL 3 Data

The following code creates the LEVEL 3 data. The data are summarized for each YEAR, REGION and STATE.

```
/*  Level 3:  10110  */

proc sort data=in_cube.ymrs_11110;
     by year region state month;
run;
data test3m;
   set ymrs_11110;
   by year region state ;
   if last.state;
run;
proc means data=ymrs_11110 sum nway noprint;
     class year region state;
     var Available_Cars Leased_Cars Amount_Billed Amount_Collected;
     output out=test3y sum=;
run;

data in_cube.Y_RS_10110(drop=_Fre: ) ;
     merge test3y(drop=Available_Cars  Leased_Cars  _type_)
           test3m(drop=Amount_Billed   Amount_Collected _type_);
     by year region state;
     month=.;
     _Type_='10110';
run;
```

Program 4.

The pattern for this program is the same as for program 3. The first DATA step gets the last row for each STATE / REGION / YEAR. The PROC MEANS step gets the SUM for each STATE / REGION / YEAR. The last DATA step merges the two datasets together so that each row has the SUMS for AMOUNT_BILLED and AMOUNT_COLLECTED and the LAST ROW for AVAILABLE_CARS and LEASED_CARS.

The resulting dataset is shown in the PROC PRINT output below.

year	month	Region	State	city	Amount_ Billed	Amount_ Collected	Available_ Cars	Leased_ Cars	_Type_
2008	.	Central	KS		255,200	203,869	258	233	10110
2008	.	Central	MI		103,100	83,386	114	107	10110
2008	.	Central	TX		172,800	135,223	209	157	10110
2008	.	East	FL		276,700	219,327	274	244	10110
2008	.	East	NC		233,400	190,797	288	231	10110
2008	.	East	NY		223,400	177,897	267	240	10110
2008	.	West	CA		225,600	183,892	221	203	10110
2008	.	West	OR		214,300	171,909	214	198	10110
2008	.	West	WA		149,900	121,194	155	149	10110
2009	.	Central	KS		260,200	203,782	258	228	10110
2009	.	Central	MI		102,400	79,140	114	107	10110
2009	.	Central	TX		209,800	169,987	209	196	10110
2009	.	East	FL		283,100	226,362	274	254	10110
2009	.	East	NC		262,400	217,724	288	225	10110
2009	.	East	NY		272,200	217,387	267	253	10110
2009	.	West	CA		228,200	187,436	221	209	10110
2009	.	West	OR		216,200	166,639	214	197	10110
2009	.	West	WA		152,600	117,846	155	151	10110
					=========	=========	==========	=======	
					3,841,500	3,073,797	4,000	3,582	

Figure 6: PROC PRINT Output.

By using the SUM statement in PROC PRINT, the 2 additive columns 'add up' to match the totals for the entire dataset.   But, when we add the NON-Additive columns (Available_Cars and Leased_Cars) do not match the totals for the entire dataset.

LEVEL 4 Data

The following code creates the LEVEL 4 data. .The data are summarized for each YEAR, REGION, STATE and MONTH.

```
/*  Level 4:  10110  */

proc sort data=in_cube.ymrsc_11111;
     by year region state city month;
run;

data test4m;
   set in_cube.ymrsc_11111;
   by year region state city ;
   if last.city;
run;

proc means data=in_cube.ymrsc_11111 sum nway noprint;
     class year region state city;
     var Available_Cars Leased_Cars Amount_Billed Amount_Collected;
     output out=test4y sum=;
run;

data in_cube.Y_RSC_10111(drop=_Fre: ) ;
     merge test4y(drop=Available_Cars  Leased_Cars  _type_)
           test4m(drop=Amount_Billed   Amount_Collected _type_);
     by year region state;
     month=.;
     _Type_='10111';
run;
```

.Program 5: Generating LEVEL 4 Data.

We now have the following datasets:

Dataset Name	Level	Rows

- **ALL_00000** - Level 0 - 1 row.
- **Y_10000** - Level 1Y - 2 rows – 1 per Year.
- **YM_11000** - Level 1M - 24 rows – 1 per Year per Month.
- **Y_R_10100** - Level 2Y - 6 rows – 1 per Year per Region.
- **YMR_11100** - Level 2M - 72 rows – 1 per Year / Month / Region.
- **Y_RS_10110** - Level 3Y - 18 rows – 1 per Year / Region / State.
- **YMRS_11110** - Level 3M - 216 rows – 1 per Year / Month / Region / State.
- **Y_RSC_10111** - Level 4Y - 72 rows – 1 per Year / Region / State / City.
- **YMRSC_11111** - Level 4M - 864 rows – 1 per Year / Month / Region/State/City.

The next step is to 'Register' the data so that we can build a cube. The registration of the datasets is done in SAS Management Console and will be illustrated in the presentation.

BUILDING THE CUBE

The first PROC OLAP step deletes the CUBE if it exists. The beginning of the second step is shown here.

```
libname in_cube 'c:\Olap_cube\Cars\Data ';

proc olap delete_physical  cube=Car_Lease;
      METASVR host="localhost"  port=8561
              protocol=bridge
              userid="sasdemo"  pw="sasbtc"
              repository="Foundation"
              olap_schema="SASMain - OLAP Schema";
run;


PROC OLAP cube=Car_Lease
            path="c:\Olap_Cube\Cars"
            description="Car Lease Cube" ;

      METASVR  host="localhost"  port=8561   protocol=bridge
               userid="sasdemo"  pw="sasbtc"
               repository="Foundation"
               olap_schema="SASMain - OLAP Schema";
```

Program 6. PROC OLAP step

The first **DIMENSION** statement is shown. The TIME DIMENSION contains the TIME HIERARCHY which has the LEVELS Year and Month.

```
DIMENSION Time  hierarchies=(Time)
          CAPTION = 'Time Dimension'
          SORT_ORDER = ASCENDING ;

    HIERARCHY Time
    levels=(Year Month)
    CAPTION='Time Hierarchy' ;

        LEVEL Year
        CAPTION='Year'
        SORT_ORDER=ASCENDING
        ;

        LEVEL Month
        CAPTION='Month'
        SORT_ORDER=ASCENDING
        ;
```

Program 7. The DIMENSION Statement and the HIERARCY Statement for TIME.

The next DIMENSION statement is for the GEOGRAPHY dimension.

```
DIMENSION Geography  hierarchies=(Geography)
    CAPTION='Geography Dimension'
    TYPE=GEOGRAPHY  SORT_ORDER=ASCENDING ;

    HIERARCHY Geography /* ALL_MEMBER='All Geography' */
    levels=(Region  State  City)
    CAPTION='Geography Hierarchy'  DEFAULT ;

    LEVEL Region
    CAPTION='Region'
    SORT_ORDER=ASCENDING
    ;

    LEVEL State
    CAPTION='State '
    SORT_ORDER=ASCENDING
    ;

    LEVEL City
    CAPTION='City'
    SORT_ORDER=ASCENDING
    ;
```

Program 8. The DIMENSION Statement and the HIERARCY Statement for GEOGRAPHY

The four MEASURE statements come next.

```
MEASURE Amount_Billed_Sum
         STAT=SUM
         Aggr_COLUMN=Amount_Billed
         CAPTION='Sum of Amount Billed'
         FORMAT=Dollar12.2
         /*DEFAULT*/
         ;
MEASURE Amount_Collected_Sum
         STAT=SUM
         Aggr_COLUMN=Amount_Collected
         CAPTION='Sum of Amount Collected'
         FORMAT=Dollar12.2
         /*DEFAULT*/
         ;
MEASURE Available_Cars
         STAT=SUM
         Aggr_COLUMN=Available_Cars
         CAPTION='Available Cars'
         FORMAT=comma8.
         /*DEFAULT*/
         ;
MEASURE Leased_Cars
         STAT=SUM
         Aggr_COLUMN=Leased_Cars
         CAPTION='Leased Cars'
         FORMAT=comma8.
         /*DEFAULT*/
         ;
```

Program 9. The MEASURE Statements.

Each MEASURE statement names the numeric variable that is the measure. It names the statistic as well as the CAPTION which is basically a label.

Next in the program are the AGGREGATION statements. They specify the aggregation and name all the variables that make up the granularity of that aggregation. Following the '/' the table= option is specified that names the dataset that will be used for a specific aggregation level. This is where the magic really occurs. In 'traditional' drilldown hierarchies, as you drill from one level to the next, you are going from one VARIABLE to another in the SAME dataset. With NON – Additive measures, as you drilldown, you go from one DATASET to another. The dataset for a specific aggregation level is named on the AGGREGATION statement.

```
AGGREGATION year month region state city
            / table=in_cube.YMRSC_11111  NAME='Level 4M'
            ;

AGGREGATION year region state city
            / table=in_cube.Y_RSC_10111  NAME='Level 4Y'
            ;

AGGREGATION year month region state
            / table=in_cube.YMRS_11110  NAME='Level 3M'
            ;

AGGREGATION year region state
            / table=in_cube.Y_RS_10110  NAME='Level 3Y'
            ;

AGGREGATION year month region
            / table=in_cube.YMR_11100  NAME='Level 2M'
            ;

AGGREGATION year region
            / table=in_cube.Y_R_10100  NAME='Level 2Y'
            ;

AGGREGATION year month
            / table=in_cube.YM_11000  NAME='Level 1M'
            ;

AGGREGATION year
            / table=in_cube.Y_10000  NAME='Level 1Y'
            ;
```

Program10. The AGGREGATION Statements.

The only thing after the AGGREGATION statements is the RUN statement. When this above PROC OLAP step is submitted, the CAR_LEASE 'cube' is built.

During the presentation of this paper, the cube will be surfaced in Enterprise Guide as well as Web Report Studio.

CONCLUSION

By knowing how to write PROC OLAP code, you can create ‘cubes’ that might not be creatable through OLAP Cube Studio.

RECOMMENDED READING

There are a number of books and training courses available from SAS Institute, Inc. Some courses of interest might be: Creating and Viewing OLAP Cubes; Overview of SAS Business Intelligence and Data Integration Applications; What’s New in SAS 9.2 Business Intelligence; Designing, Tuning and Maintaining OLAP Cubes; and SAS OLAP Environment Administration.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Name:	Ben Cochran
Company:	The Bedford Group
Address:	3224 Bedford Ave.
City, State ZIP:	Raleigh, NC
Work Phone:	(919) 741-0370
E-mail:	bedfordgroup@nc.rr.com