SAS®9 OLAP Server: Data Management for ROLAP Aggregations

Table of Contents

Introduction: Cubes as collection of summarized data

Queries against OLAP cubes are fast because cubes access summarized data. The more pre-summarized aggregations a cube has, the faster it can answer queries. A well-tuned cube is a cube that has a well-chosen set of aggregations.

SAS OLAP cubes allow you to either create those pre-summarized aggregations as part of the cube itself (we call these MOLAP aggregations), or to point to external data (this is called ROLAP data access).

When you point your cube to external data, you have two options:

1. Point to the input data only, without aggregations (NO_NWAY)
   This is a good option if your input data are on a very fast server, and summarization on-the-fly can be done just as fast as reading the summarized data.
   This may also be your choice if your external database has its own aggregation table management.
2. Point to external summarized tables (ROLAP aggregations)

This note will talk in particular about the last option - using ROLAP aggregations in your SAS OLAP Cube.

When you use ROLAP aggregations, you can use any database format that is accessible by the SAS System. SAS itself has a number of options (e.g. SAS datasets, SPDE datasets, SPDS, TKTS Table server), and the extensive range of SAS/ACCESS products that allow access to practically any database available.

When creating or accessing existing ROLAP aggregations, you want to take a few considerations into account to make sure your aggregation tables work correctly and efficiently with the SAS OLAP Server.

What is an aggregation table?

First, a couple of definitions:

- When you define a cube, you specify hierarchies with **Levels**. Each level corresponds to a column in your input data. In the ROLAP aggregation table, we call these **level columns**.

- In your cube, you also define **Measures**. Each measure corresponds to an input column combined with a roll-up-rule (also known as "statistic"). In the aggregation table, we call these **measure columns**.

These definitions allow us to give the following definitions:

- An aggregation table always consists of **level columns** and **measure columns**.
- All aggregation tables for a cube have the **same** set of **measure columns**.
- All aggregation tables for a cube have a **different** set of **level columns**.

Measure Columns: Stored Statistics vs. Derived Statistics

Not all statistics need to be represented by a column stored in the aggregation tables.
SAS OLAP cubes retrieve only the following six statistics from aggregation tables:

SUM, N, NMISS, USS, MIN, MAX

The OLAP Server derives the other available statistics from the stored statistics. For example, in order to include a measure for the AVG statistic in your cube, you need to make columns available in your aggregation tables that were generated by using SUM and COUNT (or N).

Here is the full list of Derived Statistics:

Derived Statistic	Required Stored Statistic
AVG	N, SUM
CSS	N, SUM, USS
RANGE	MIN, MAX
VAR, STD, STDERR, CV, T, PRT, LCLM, UCLM	N, SUM, USS

Example

Consider the following structural elements in this simple cube with 5 levels, 3 measures and 2 ROLAP aggregations. NOTE: The following code is will not create a functional cube. It contains those elements of the code relevant to this example. Other dimensions and aggregations could exist.

```
PROC OLAP CUBE=CarsCube DATA=rtables.cars …;

…;

DIMENSION Date    HIERARCHIES=(Date)       SORT_ODER=ASCENDING;
HIERARCHY Date    LEVELS=(dte);

DIMENSION Cars    HIERARCHIES=(Cars)       SORT_ORDER=ASCENDING;
HIERARCHY Cars    LEVELS=(car color);

DIMENSION Dealers HIERARCHIES=(Dealers)    SORT_ORDER=ASCENDING;
HIERARCHY Dealers LEVELS=(dealer dest);

MEASURE SalesSum COLUMN=sales STAT=SUM FORMAT=DOLLAR15.2 AGGR_COLUMN=sales_sum;
MEASURE SalesN   COLUMN=sales STAT=N   FORMAT=12.0       AGGR_COLUMN=sales_n;
MEASURE SalesAvg COLUMN=sales STAT=AVG FORMAT=10.0;

AGGREGATION car dealer     / TABLE=rtables.cars_d1_d2;
AGGREGATION car            / TABLE=rtables.cars_d1;

...
```

Dte	Car	Color	Dealer	Dest	Sales
January	Toyota	Red	Smith	NC	10000
February	Toyota	Red	Smith	CT	15000
March	Chevy	Green	Smith	NJ	17000
April	Ford	Blue	Smith	CA	12000
May	Toyota	Red	Jones	NC	4000

Table 1: Input Table rtables.cars

Car	Dealer	Sales_sum	Sales_n
Chevy	Smith	17000	1
Ford	Smith	12000	1
Toyota	Jones	4000	1
Toyota	Smith	25000	2

Table 2: Aggregation table rtables.cars_d1_d2

One way to create the above table from the input table would be an SQL SELECT statement:

```
create table rtables.cars_d1_d2 as
select car, dealer,
       sum(sales) as sales_sum,
       count(sales) as sales_n
from rtables.cars
group by 1, 2
```

Car	Sales_sum	Sales_n
Chevy	17000	1
Ford	12000	1
Toyota	29000	3

Table 3: Aggregation table rtables.cars_d1

Another way to create an aggregation is by using the SAS MEANS procedure:

```
PROC MEANS NOPRINT NWAY MISSING DATA=olapsio.cars;
CLASS car ;
OUTPUT OUT=cars_d1(DROP=_FREQ_ _TYPE_)
   SUM(sales) = sales_sum
   N(sales) = sales_n
;
RUN;
```

Notice the following:

- The `sales` column in the input table becomes the **measure columns** `sales_sum` and `sales_n` in the aggregation tables. The names of the measure columns are specified in the `AGGR_COLUMN` option in the `MEASURE` statements.
- There is no measure column for the `SalesAvg` measure. The value for that measure is derived at OLAP server runtime from the measures `SalesSum` and `SalesN.`
- The list of level columns in the `AGGREGATION` statements matches the list in the `GROUP BY` clause (or the `CLASS` statement) of the corresponding aggregation table creation jobs.
- The measure columns in the both the aggregation tables are the same.

Important considerations

What follows are a number of important things to keep in mind when using ROLAP aggregation tables.

Attribute Consistency

- The attributes of the **level columns** need to be the same in the input data and the aggregation tables. In particular, the following attributes need to match:

Name	Length	Type	SAS Format

- The names of the **measure columns** need to match the names given in the AGGR_COLUMN option (on the MEASURE statement).

Data Integrity

The cube input data and the aggregation tables need to be consistent with each other. It is vital that the data in the aggregation tables stay in synch with the cube input data. An aggregation table always needs to be created from all input data (and not a subset of the input data). If your aggregation tables are not based on the same data as your cube, the data values that the users see in their reports may be incorrect, and in many cases, users will run into errors like the one described here: support.sas.com/kb/13/289.html.

Implicit Pass-Through

One of the benefits of using ROLAP cubes is that it allows you to keep data access and processing on the source database. Any summarization on-the-fly is being performed on the RDBMS, and only the (typically small) results are being read by the OLAP server.

The SAS/ACCESS engines use Implicit Pass-Through to push the processing into the database. If the processing for some reason cannot be passed to the database, you will often see a considerable performance impact, because all the data necessary for the processing is being transferred into the SAS System.

When building ROLAP SAS OLAP cubes, there are two scenarios that can disable Implicit Pass-Through:

Using SAS Formats

Using SAS Formats on level columns can be an efficient way to save space in your input tables. But in cases where your SAS Format groups multiple values into one (for example individual dates formatted into month names), the WHERE clause of the SQL that is being generated to retrieve data from your ROLAP aggregations may need to apply that FORMAT to your level column, using the SAS PUT() function. This can adversely affect performance, and may disable Implicit Pass-Through.

Many-to-one formats can also lead to problems with unexpected member ordering. See support.sas.com/kb/19/651.html for more detail.

Note that SAS Formats on measure columns have no effect on data retrieval. They are being applied at display time only

Dimension Names with special characters or reserved words

Dimension names are being used as table aliases in the SQL that is being generated to retrieve data from your ROLAP aggregations. If the names have special characters, blanks, for reserved words (e.g. TIME), it can break Implicit Pass-Through. See support.sas.com/kb/40/911.html for more detail.

Creating multiple levels from the same column

SAS OLAP Cubes in 9.2 allow you to create multiple levels from just one input column. See the FORMAT= and COLUMN= options on the PROC OLAP LEVEL statement: support.sas.com/documentation/cdl/en/olapug/59574/HTML/default/viewer.htm#a002608965.htm for more detail.

When you use that feature you won't be able to use ROLAP aggregation tables because the cube build process requires a one-to-one match between level and input column.

Conclusion

Using your own aggregation tables in your SAS OLAP Cubes allows for flexible data management and fast cube creation. With some attention to the data preparation you can help make sure that the cube works well with your aggregation tables. Give special attention when using numeric categories, many-to-one formats, dimension names with non-standard characters, and multiple formats on the same input column.

References

SAS® 9 OLAP Server: Cube Building – Data Preparation and Cube Storage
(support.sas.com/rnd/olap/cubebuilding2_042804.pdf)

Using SAS® OLAP Server for a ROLAP Scenario
(support.sas.com/resources/papers/proceedings09/103-2009.pdf)

Using SAS Formats in Teradata
(support.sas.com/documentation/cdl/en/acreldb/63647/HTML/default/viewer.htm#a003276900.htm)

Netezza (support.sas.com/documentation/cdl/en/acreldb/63647/HTML/default/viewer.htm#a003331724.htm)

SAS Institute Inc. World Headquarters +1 919 677 8000

To contact your local SAS office, please visit: **sas.com/offices**