SAS® 9.4 OLAP Server: User's Guide, Fifth Edition

SAS® Documentation

October 20, 2022

The correct bibliographic citation for this manual is as follows: SAS Institute Inc. 2016. *SAS® 9.4 OLAP Server: User's Guide, Fifth Edition*. Cary, NC: SAS Institute Inc.

SAS® 9.4 OLAP Server: User's Guide, Fifth Edition

SAS Institute Inc., SAS Campus Drive, Cary, NC 27513-2414

January 2023

9.4-P7:olapug

Contents

1

OLAP Introduction and Overview

OLAP Introduction and Overview

What Is OLAP?

Online Analytical Processing (OLAP) is a technology that is used to create decision support software. OLAP enables application users to quickly analyze information that has been summarized into multidimensional views and hierarchies. By summarizing predicted queries into multidimensional views prior to run time, OLAP tools provide the benefit of increased performance over traditional database access

tools. Most of the resource-intensive calculation that is required to summarize the data is done before a query is submitted.

Data Storage and Access

Decision makers are asked to make timely and accurate decisions that are based on the past performance and behavior of an organization as well as on future trends and directives. To make effective business decisions, business analysts must have access to the data that their company generates and responds to. This access must include timely queries, summaries, and reviews of numerous levels and combinations of large, recurrent amounts of data. The information that business analysts review determines the quality of their decisions.

Organizations usually have databases and data stores that maintain repeated and frequent business transaction data. These data storage systems provide simple yet detailed storage and retrieval of specific data events. However, these systems are not well suited for analytical summaries and queries that are typically generated by decision makers. For decision makers to reveal hidden trends, inconsistencies, and risks in a business, they must be able to maintain a certain degree of momentum when querying the data. An answer to one question usually leads to additional questions and review of the data. Simple data stores do not successfully support this type of querying.

A second type of storage, the data warehouse, is better suited for this. Data is maintained and organized so that complicated queries and summaries can be run. OLAP further organizes and summarizes specific categories and subsets of data from the data warehouse. This results in a robust and detailed level of data storage with efficient and fast query returns. SAS OLAP cubes can be built from either partially or completely denormalized data warehouse tables. Stored, precalculated summarizations called aggregations can be added to the cube to improve cube access performance. Aggregations can either be pre-built relational tables, or you can let the cube create its own optimized aggregates.

Benefits of OLAP

The ability to have coherent, relevant, and timely information is the reason OLAP has gained in popularity. OLAP systems can help reveal evasive inconsistencies and trends in data that might not have been seen before. OLAP users can intuitively search data that has been consolidated and summarized within the OLAP structure. In addition, OLAP tools allow for tasks such as sales forecasting, asset analysis, resource planning, budgeting, and risk assessment. OLAP systems also provide the following benefits:

- fast access, calculations, and summaries of an organization's data
- support for multiple user access and multiple queries
- the ability to handle multiple hierarchies and levels of data
- the ability to presummarize and consolidate data for faster query and reporting functions
- the ability to expand the number of dimensions and levels of data as a business grows

Implementation

To fully understand the benefits of OLAP and the details of its effective implementation, it helps to examine the technology from two perspectives— that of the users, and that of the information technology (IT) administrators who are responsible for OLAP implementation. The users, typically business analysts and executives, expect the data to be organized according to categories that reflect the way they think about the enterprise. For IT administrators, OLAP can present a long list of technical issues, including these concerns:

- storage requirements and associated costs
- client and server capabilities
- maintenance activities such as update and backup
- performance considerations such as the amount of time that is required to build a multidimensional model
- the ability of the OLAP solution to integrate with current or planned data warehouse strategies and architectures
- security requirements for cube data

The SAS OLAP Server and SAS OLAP Cube Studio provide resources and functionality to address these concerns. When building SAS OLAP cubes, you can perform functions and specify settings that affect the following:

- cube aggregation storage and query performance
- cube dimension security and identity-driven security
- updates of cube data
- maintenance of cubes (such as adding calculated members or changing an OLAP schema.)

Because SAS OLAP is a component of the SAS Intelligence Platform, it works in conjunction with other SAS applications to provide an overall solution to the access and maintenance of a company's data.

What Is a Cube?

One of the advantages of OLAP is how data and its relationships are stored and accessed. OLAP systems house data in structures that are readily available for detailed queries and analytics. Cubes are central to the OLAP storage process.

A cube is a set of data that is organized and structured in a hierarchical, multidimensional arrangement. The cube is usually derived from a subset of a data warehouse. Unlike relational databases that use two-dimensional data structures (often in the form of columns and rows in a spreadsheet), OLAP cubes are logical, multidimensional models that can have numerous dimensions and levels of data. Also, an organization typically has different cubes for different types of data.

One of the challenges of OLAP cube data storage and retrieval is the growth of data and how that growth affects the number of dimensions and levels in a cube hierarchy. As the number of dimensions increases over time, so does the number of data cells on an exponential scale. To maintain the efficiency and speed of the OLAP queries, the cube data is often presummarized into various consolidations and subtotals (aggregations).

Note: The SAS OLAP Server term cube is synonymous with the terms hyper-cube and multi-cube.

Understanding the Cube Structure

Cube Structure

A SAS OLAP cube stores data in a method that enables fast retrieval of summarized data. Data summarization in this context means condensing large numbers of detail records into meaningful numbers such as counts, sums, averages, or other statistical measures. The structure of a cube is hierarchical in nature and is derived from the associations between the different columns and rows of data in a data source. SAS OLAP cubes consist of dimensions, levels, hierarchies, members, and member properties. This structure enables you to easily select data subsets and navigate the cube structure when querying the cube.

Dimensions, Levels, and Hierarchies

SAS OLAP cubes organize data in a hierarchical arrangement, according to dimensions and measures. Dimensions group the data along natural categories and consist of one or more levels. Each level represents a different grouping within the same dimension. For example, a time dimension can include levels such as years, months, and days. Or an organization dimension of a bank's customer service centers can include levels such as branches, states, and regions.

Levels are organized into one or more hierarchies, typically from a coarse-grained level (for example, Year) down to the most detailed one (for example, Day). The individual category values (for example, 2002 or 21Jan2002) are called members.

A dimension can also have multiple hierarchies to provide different sequences of groupings. For example, a "Time" dimension can have a "Fiscal Year" hierarchy and a "Calendar Year" hierarchy.

Members

Each combination of values within a dimension is called a member. Some examples of members are shown here.

- [Time].[2003]
- [Time]. [2004].[January]
- [Time]. [2004].[February].[12th]

For each dimension, there is also the special member called the ALL member, which represents the total for all members (for example, [Time].[All Time]). Not all categorical data attributes need to become a member of a hierarchy level. Some grouping information is needed only as additional information for a member or for applying subsets to data. These attributes can be loaded into member properties. Member properties can be associated with any level in a hierarchy.

Measures

Measures are the cube data values that are summarized and analyzed. A measure is the combination of a numeric input column with a roll-up rule or statistic. Measures are loaded from the data source that you summarize from. One input column can load one or more measures. For example, you can create the measures "Sum of Amount" and "Maximum of Amount" from the input column "Amount."

Calculated Measures

Not all measures are directly derived from input columns. You can create calculated measures, which are formulas that are based on the values of other measures. Calculated measures are designed and stored with the cube.

Aggregations

An aggregation is a summary of detail data that is stored with or referred to by a cube. They are the basis for fast response to data queries in OLAP applications. An aggregation is possible at each intersection of a level of one or more dimensions. Any combination of dimension levels can become a stored aggregation, as long as it is appropriate within the defined hierarchies. One of the major factors that influences query response time is which aggregations you create and use to query your cube. The aggregations that are being stored with the cube affect cube build time, the absolute cube file size, SAS OLAP Server CPU usage, and query response times. As a result, determining and building your cube aggregations is a crucial component of good cube design.

Why You Should Use Cubes

Cube Usage and Storage Space Reduction

SAS cubes are designed to offer efficient data storage, fast data access, easy data maintenance, and flexibility in data management. While cubes are the format of choice to guarantee fast query response times against your data warehouse, SAS OLAP cubes are also often a very space-efficient choice for data storage. In many cases, a basic cube without additional aggregations can be smaller than the input data because the process of creating the cube consolidates records. SAS OLAP cubes use the hierarchy information for efficient aggregations storage. SAS OLAP cubes also deal efficiently with data sparsity by using virtual placeholders for empty cells. This removes the need for any physical representation of empty cells. A good rule of thumb is, the larger your input data, the greater the storage gain by loading data into a cube.

Multi-Threading Capabilities

Loading data into cubes and executing queries against the cube take advantage of the multi-threading capabilities of your server machine. Aggregations are created in parallel at cube build time. The creation of individual aggregations takes advantage of the Parallel Group-By capabilities of the SAS data engine. At query execution, the multi-threading capabilities of your server machine are fully used to concurrently serve queries by multiple users. Both query evaluation and data access are executed in parallel.

Easy Setup and Maintenance

A cube is the physical representation of your logical dimensional model. The tools that are provided to update and maintain the cube reflect the multidimensional model, which makes both setup and maintenance of your cube as intuitive as possible. SAS Management Console, a Web-based administrator interface, enables you to set up and manage OLAP servers. SAS OLAP Cube Studio provides the workspace and cube designer tools that you need to create and maintain cubes. You can also use the SAS OLAP procedure to create and maintain cubes in a batch environment.

Data Management: Choosing Your Own Tool

If you create your own aggregations by using data management tools such as SQL, PROC SUMMARY, or the tools of your preferred relational database management system (RDBMS), then you can link those aggregations to your cubes without replicating the data within the cube. Any queries against those aggregations are executed by the appropriate SQL engine, and take advantage of any capabilities that engine might have. This allows you the flexibility to use the data management tools of your choice. It also enables you to distribute your data for your cube aggregations across multiple database systems, servers, and platforms. If you choose to let the cube builder create the aggregations, then you can control where to store the data and index files for each aggregation.

2

SAS OLAP Variations

SAS OLAP Variations

Overview

OLAP technology can be further defined by the methods for storing and accessing data and by the performance of queries against that data. A SAS cube consists of three parts: metadata, navigation files, and the physical data or aggregation tables. Metadata and navigation files are consistent across the different OLAP techniques. The metadata for a cube defines information such as the location of data, the cube structure, cube-based security permissions, and calculated measure definitions. The navigation files are used to determine how the input data information translates to the structure of the cube. For example, the navigation files determine the following:

- how cube members relate to each other
- what formats are used
- what the member properties are
- what the captions for each member are

The physical data (aggregation tables) is dependent on which OLAP technique the cube designer specifies when building the cube structure. SAS OLAP offers a flexible OLAP solution, not only allowing users to choose a multidimensional OLAP (MOLAP) approach, but also a relational OLAP (ROLAP) solution. SAS OLAP supports three different variations of OLAP technology.

- MOLAP (multidimensional OLAP)
- ROLAP (relational OLAP)
- HOLAP (hybrid OLAP)

MOLAP (multidimensional OLAP)

MOLAP is a type of OLAP that stores summaries of detail data (aggregates) in multidimensional database structures. MOLAP cubes are most suitable for slicing and dicing of data and are used when performance and query speed is critical.

With a MOLAP approach, relevant SAS proprietary highly indexed aggregation tables are created and stored within the physical cube. Since the data is pre-aggregated, the response is quickly returned to the end users. Although it is a popular technique that provides quick access to detail data, MOLAP introduces a set of challenges.

One of these challenges is a cost challenge related to the overhead of the tasks the administrators must perform. In a MOLAP structure, the data resides in both the data warehouse and the cube. This means that the data must be updated and maintained in two locations.

When building a MOLAP cube, the majority of time is spent in transferring data and processing aggregations. As businesses expand their analysis to include more dimensions or deeper levels of analytics, the cost and overhead of moving and replicating data into an external cube becomes a challenge for the administrator.

The common workaround is to reduce the cube size (amount of data to be transferred). However, this results in less data for analysis, fewer dimensions and details, and reduced value for business analysis. Reduction in cube update frequency is also a feasible option but leads to OLAP cubes periodically becoming out-of-sync with the data warehouse. This can result in less accurate analytics.

For further information about MOLAP, see the topic “MOLAP Aggregation Storage” on page 48.

ROLAP (relational OLAP)

To meet the challenges introduced by MOLAP, another technique can be used by customers. ROLAP is a type of OLAP in which multidimensional data is stored in a relational database such as a SAS table or an ORACLE table. ROLAP is more scalable than other OLAP structure types and handles extensive amounts of data well. A ROLAP approach reduces cube build times and decreases maintenance activities as the data remains in relational database management systems (RDBMS). Although performance can be somewhat slow, ROLAP is limited only by the size of the relational database it is identified with.

With a ROLAP approach, the SAS OLAP Server accesses data stored within relational database management systems directly. All data resides in the RDBMS, where relational tables are optimized for low-level dimensional requests, and aggregate indexes are created for higher-level OLAP requests. A ROLAP-based cube allows the RDBMS to handle the SQL code and optimization.

The advantages of a ROLAP solution include the following:

- Only metadata and navigation files are created, resulting in fast build times. ROLAP aggregations build in a fraction of the time it takes to build a MOLAP cube. Data expansion results in minimal impact on cube build time.
- The warehouse updates are immediately available for users who query the cube.
- Data management remains within the RDBMS, not within the cube.

The physical database design of any data warehouse should reflect the customer's business, independent of any tool or application requirements. A normalized data model, snowflake schema, and star schema are all widely used as data models within enterprises.

SAS OLAP cubes support three types of input data: star schema, detail data, or summarized tables. When SAS ROLAP is implemented on a normalized data model or snowflake schema, an additional semantic layer must be built on top of the table to represent a star schema. It is crucial that the normalized model or snowflake schema be cleansed. Cleansing data refers to manipulating the data so that data transformations are complete, NULLs are absent, and data is ready for reporting.

If the data is not cleansed, it might be necessary to build a physical semantic layer. If a physical semantic layer is required, it is recommended to implement a snowflake schema that is populated by INSERT and SELECT from the normalized model in the database. The INSERT and SELECT options would be defined so that they perform the data-cleansing tasks to result in a snowflake schema that is ready for reporting. A SAS cube that feeds from a star schema representation of DBMS data should be defined with the 'Do not create an NWAY' option. This fully summarized table, composed of all crossing of the levels defined in the cube, is equivalent to the PROC OLAP option NO_NWAY.

For further information about ROLAP, see the topic "ROLAP Aggregation Storage" on page 48

HOLAP (hybrid OLAP)

HOLAP is a type of OLAP in which relational OLAP (ROLAP) and multidimensional OLAP (MOLAP) are combined. In HOLAP, the source data is usually stored using a ROLAP strategy, and aggregations are stored using a MOLAP strategy. It combines the best features of both ROLAP and MOLAP. This combination usually results in the smallest amount of storage space. In HOLAP, aggregates can be precalculated and can be linked into a hybrid storage model.

The HOLAP technique addresses some of the challenges of the MOLAP implementation. With a HOLAP approach, a mix of the SAS proprietary aggregation tables and relational tables is used. This provides the business intelligence administrator the flexibility to establish the location of the multi-dimensional data. The location choice depends on the access frequency, administration, and processing overhead of the data. The comparative performance and scalability benefits or penalties between the ROLAP, MOLAP, or HOLAP approaches would have to be determined for any given application.

ROLAP Implementation and the SAS OLAP SERVER

SAS OLAP uses a combination of SAS servers to store cube metadata, to store the physical cube structure, and to query cubes after they are created. Several types of SAS servers are available to handle different workload types and processing intensities. The SAS OLAP Server is a scalable server that provides multi-user access to data that is stored in SAS OLAP cubes or populated in real time from relational databases. This server is designed to reduce the load on traditional back-end storage systems by quickly delivering summarized views, regardless of the amount of data that underlies the summaries. OLAP queries are performed by using the Multidimensional Expressions (MDX) query language in client applications that are connected to the SAS OLAP server.

The SAS OLAP Server has a dual role. It provides security validation and a query engine. Security validation includes the following tasks:

- authentication of the user against the SAS Metadata Server
- authorization and validation of what the user is allowed to see

The OLAP technique that the cube is based on determines how the MDX query is handled and translated into an appropriate query. This query is passed either to an underlying database or processed internally. In the MOLAP-based cube, the SAS OLAP Server spawns multiple threads internally to retrieve the queries from the relevant cube aggregation tables. For a ROLAP-based cube, MDX is translated into SQL queries, which are passed down to the RDBMS to handle and optimize.

For the most optimized performance at query time, a SAS cube requires aggregation tables that best meet the query result set. SAS provides Application Response Measurement (ARM) logs that help cube designers or administrators tune a cube based on end-user interactions or queries that have been submitted against that cube.

A cube designer can define a cube using SAS OLAP Cube Studio or using procedure code (PROC OLAP). To build a cube by using either of those methods, you must complete several preliminary tasks:

Define a SAS OLAP Server and OLAP schema	The server does not need to be running to create cubes, but it must be defined in the metadata. A SAS OLAP schema is an organization container for SAS OLAP cubes. It communicates with a SAS OLAP Server about which cubes can be accessed by the server and then queried. A SAS OLAP Server can be associated with only one OLAP schema.
Define a DBMS server	Your library needs to know which data sources to take data from. Defining a DBMS server and assigning it to the library serves this purpose.
Define a SAS library	For SAS OLAP, libraries serve as organization containers for the data tables that you use to create SAS OLAP Cubes.
Register source data tables	In order to register data tables, you must have a SAS Workspace Server registered in the metadata and running while registration takes place.

Different tools are available to complete the preceding steps, including SAS Management Console.

In addition to these preliminary tasks, proper authorization permissions should be assigned to users in order for them to complete the metadata setup and other tasks. These permissions must meet the following requirements:

- ReadMetadata and WriteMetadata permissions must be assigned to the user who establishes the metadata.
- Users who will query ROLAP cubes must have the set of security permissions assigned based on the data (from the cube) they are allowed to access. The same users should be defined as members of the Authentication Domain that was specified for your DBMS server.
- Users who will query ROLAP cubes should be also defined on the DBMS side. They should have select privileges granted on objects of the physical schema that the cube is built upon.

3

Defining the SAS OLAP Environment

SAS OLAP Environment — Overview

Installation Planning

Before you can begin developing SAS OLAP cubes, you must define and install the servers, tools, and user interfaces that enable you to create and manage your SAS OLAP cubes and related data. When you install your SAS OLAP environment, you will need to plan for and include the following items:

SAS Servers

SAS OLAP uses a combination of SAS servers to store cube metadata, to store the physical cube structure, and to query cubes after they are created. The term server refers to a program or programs that wait for and fulfill requests from client programs for data or services. Several types of SAS servers are available to handle different workload types and processing intensities.

- a “SAS Metadata Server” on page 17
- a “ SAS Application Server” on page 18
- a “ SAS Workspace Server ” on page 18
- a “SAS OLAP Server” on page 18

SAS Clients

The SAS OLAP environment uses the following Java desktop applications:

- “SAS Management Console” on page 25
- “SAS OLAP Server Monitor” on page 25
- “SAS OLAP Cube Studio” on page 25

Data Organization and Management Components

The following data organization and management components are necessary for and used by the SAS OLAP environment.

- “OLAP Schema” on page 26

User and User Group Authorization Permissions

SAS Metadata Server

The SAS Metadata Server is a multi-user server that enables users to read metadata from and write metadata to one or more SAS metadata folders. This server is a centralized resource for storing, managing, and delivering metadata for all SAS applications across the enterprise. It enables all users to access consistent and accurate data. Some types of metadata objects that can be stored in the SAS metadata include data libraries, tables, cubes, users and user groups, user authorization permissions, and server definitions. The default metadata server that is installed during your system installation and configuration is the SASMeta server.

You can define and administer SAS Metadata Servers in SAS Management Console. The Server Manager and Metadata Manager plug-ins contain various administrative functions. The Server Manager provides functions to manage SAS server definitions stored in the SAS Metadata. It enables you to define and edit information about server locations and connections. The Metadata Manager plug-in enables you to maintain and define the various SAS metadata for your SAS environment.

Note: For more information about the SAS Metadata Server, see the topic "SAS Metadata Server" in the *SAS Intelligence Platform: Overview*.

The SAS Metadata Server uses the Integrated Object Model (IOM) that is provided by SAS Integration Technologies. IOM provides distributed object interfaces to Base SAS software features. It enables you to use industry-standard languages, programming tools, and communication protocols to develop client programs that access these services on IOM servers. The SAS Open Metadata Interface (OMI) is an object-oriented application programming interface (API) that interacts with the SAS Metadata Server. SAS OLAP Cube Studio is an example of an application that is compliant with the SAS Open Metadata Interface.

With the SAS OLAP Server, all relevant structural information is contained within the cube and most of it is also replicated within the SAS Open Metadata Architecture. This is done so that you can do the following:

- disassociate the cube definition process from cube creation, thus enabling you to create a cube by using its stored definition
- define and enforce security at the SAS Open Metadata Architecture level
- manage and control the data source in the centralized SAS Metadata Repository

You can find documentation about the SAS Open Metadata Architecture in the *SAS Intelligence Platform: System Administration Guide* and the *SAS Intelligence Platform: Data Administration Guide*.

SAS Application Server

During installation and setup, a metadata object is defined that represents the SAS server tier in your environment. This is called the SAS Application Server. A SAS Application Server is not an actual server that can execute SAS code submitted by clients. Rather, it is a logical container for a set of application server components, which do execute code. It is identified as SASApp in the SAS Management Console interface. For a SAS OLAP installation, a SAS Workspace Server and a SAS OLAP Server are contained in the application server.

Note: For further information about the SAS Application Server, see “Understanding the SAS Application Server” in the *SAS Intelligence Platform: Application Server Administration Guide*.

SAS Workspace Server

The SAS Workspace Server provides access to SAS software features such as the SAS language, SAS libraries, and the server file system. The SAS Workspace Servers interact with SAS by creating a server process for each client connection. Each server process (workspace) represents a Foundation SAS session. This enables client programs to access SAS libraries, perform tasks by using the SAS language, and retrieve the results.

A program called the object spawner runs on a workspace servers host machine. The spawner listens for incoming client requests and launches server instances as needed. For further information about SAS Workspace Servers, see "Understanding Workspace Server and Stored Process Servers" in the *SAS Intelligence Platform: Application Server Administration Guide*.

SAS OLAP Server

The SAS OLAP Server is a scalable server that provides multi-user access to the data that is stored in SAS OLAP cubes. The purpose of the SAS OLAP Server is to respond to queries from cube viewers. SAS OLAP queries are performed by using the Multidimensional Expressions (MDX) query language in client applications that are connected to the OLAP server by using OLE DB for OLAP (an extension of OLE

DB that is used by COM-based clients), or through a similarly designed Java interface. A SAS OLAP Server runs in the background on specially configured host computers. It processes data by using a multi-threaded kernel that enables you to take advantage of your server's parallel processing abilities.

Note: For further information about the SAS OLAP Server, see "Administering SAS OLAP Servers" in the *SAS Intelligence Platform: Application Server Administration Guide*.

Setting SAS OLAP Server Options

When executing a SAS OLAP Server you can set server options from both the command line and from within the Server Manager plug-in to SAS Management Console. In SAS 9.4, a new OLAP server feature enables you to set server options in batch mode. The OLAPCONFIG file enables you to set multiple server options when starting a SAS OLAP Server. Some options that you can set with the OLAPCONFIG file include settings for cache size, available memory, and tuple size.

The OLAPCONFIG file is an XML file that is executed with the command line option **-olapconfig**. You can specify the name and, if needed, the path of the OLAPCONFIG file. If a path is specified, the file is opened and the specified server options are set. If the OLAPCONFIG file is located in the server start up directory, that file is used to set server options. On Windows, you can use the option **-sasinitialfolder** to set the invocation directory.

Any option that is set in the OLAPCONFIG configuration file will override options specified in the metadata for the logical server. The OLAPCONFIG file also overrides settings specified in the Server Manager plug-in to SAS Management Console.

Here is an example OLAPCONFIG file:

```
<OLAPConfig>
 <MAXROWS Value="1000"/>
 <MAXCELLS Value="1000"/>
 <MAXSETSIZE Value="0"/>
 <MAXCUBECACHE Value="20"/>
 <MAXFLATNROWS Value=300000"/>
 <GROUPBYROWCACHE Value="256"/>
 <MAXSEGRATIO Value="1"/>
 <NECJMAXMULTISET Value="1000000"/>
 <THREADPOOLQRY MIN="20"/>
 <THREADPOOLQRY MAX="70"/>
</OLAPConfig>
```

The following options can be set in the OLAPCONFIG file:

CACHEEMPTYSQ
: indicates to cache the empty subquery results when this is set to 1. Normally only results that have a value are stored in the subquery cache. When this option is on, all possible subquery results are stored. This requires more memory, especially when the data in the cube is sparse. However, performance can be

improved when this option is set because, depending on the query, it could result in fewer subquery calls.

The default value is 0. The corresponding SAS Management Console option is **Cache the empty subquery results**.

Note: This is the same option as *SQCACHEEMPTY*

DEBUG
sets the *DEBUG* option. The *DEBUG* option is intended only for use in diagnosing software problems. Consult SAS Technical Support for more information about how to use the *DEBUG* option.

The default value is 0. The corresponding SAS Management Console option is **Debug**.

DT_ALLCOLS
when set to 1 in the OLAPCONFIG file, prompts the SAS OLAP server to return all columns in the drill-through table. Setting this option sends a request to the OLAP server to return data for every column in the drill-through table that a user has permission to view. This behavior occurs when *DT_ALLCOLS* is set to 1. If it is not set or is set to any value other than 1, only the columns and measure variables that correspond to the cube definition are returned.

This is similar to how drill-through tables were returned in releases prior to SAS 9.4. In SAS 9.4, only columns explicitly associated with a cube definition are explicitly selected. This is the new default behavior.

DT_MAXROWS
controls the number of rows returned for each drill-through table query. This option can be used to limit the maximum number of rows that are returned for any drill-through table query. This value should be set to the maximum number of rows that you want to be returned for any drill-through table query. This value applies to every cube on the OLAP server.

For example, if *DT_MAXROWS* is set to 1000, the number of rows returned would be either 1000 or the number of rows that would normally be returned for the query if *DT_MAXROWS* was not set, whichever is less. As soon as the limit for *DT_MAXROWS* is reached, data retrieval is halted. If *DT_MAXROWS* is set to 0 or less, all rows are retrieved. This is the default behavior.

GROUPBYROWCACHE
specifies the amount of memory allocated for parallel GROUP BY operations performed by the SPD Engine.

The default (and minimum) is 256 megabytes. The corresponding SAS Management Console option is **Memory available for group by operations**.

MAXCELLS
specifies the size of the buffer measured in cells used within the OLAP Server for accessing cell information from the results of MDX query processing. In this context, a cell represents the data at each intersection of each data set axis. A larger buffer size will decrease the number of accesses needed to retrieve the information. A smaller buffer size will decrease the time needed to execute each access.

The default value of 1,000 cells provides optimal OLAP Server performance in most cases. The corresponding SAS Management Console option is **Buffer size for cellset access object**.

MAXCUBECACHE
: specifies the maximum number of cube metadata registrations that you want the server to store in memory. The metadata contains information necessary to parse and plan an MDX query to the cube. To increase server query performance, increase the number of cubes cached. To save memory at the expense of increasing query response time, decrease the number of cubes cached. The cube cache is initially empty and cubes are cached as they are accessed. As new cubes are cached, older cubes are removed according to their usage.

  The default is 20 cube metadata registrations. The corresponding SAS Management Console option is **Maximum number of cubes in cache**.

MAXFLATNROWS
: specifies the maximum number of flattened rows allowed for flattened (two-dimensional) rowsets.

  The default is 300,000 rows. The corresponding SAS Management Console option is **Maximum number of flattened rows**.

MAXFLATRSMEM
: specifies the maximum amount of memory that can be used to process flattened (two-dimensional) rowsets. If this option is set to zero or less, then all available memory can be used to process flattened rowsets.

  The default is 268,435,456 bytes. The corresponding SAS Management Console option is **Maximum memory size for flattened rowset** .

MAXROWS
: specifies the size of the buffer measured in rows used within the OLAP Server for accessing axis rowset information from the results of MDX query processing. An axis rowset provides information about each tuple on a result set axis. A larger buffer size will decrease the number of accesses needed to retrieve the information. A smaller buffer size will decrease the time needed to execute each access.

  The default value of 1,000 rows provides optimal OLAP Server performance in most cases. The corresponding SAS Management Console option is **Buffer size for rowset access object**.

MAXSEGRATIO
: specifies a percentage used by the SPD Engine to control whether segment candidate pre-evaluation is done when evaluating a WHERE expression for processing with indexes. Some queries can benefit by limiting the pre-evaluation phase, since it can be costly in certain cases. This limit is imposed by the SPD Engine based on the segment ratio, which is defined as the number of segments that contain variable values that apply to a predicate in the WHERE expression, divided by the total number of segments.

  The default value is 75, meaning that a value should be in 75% or fewer of the index segments to include the pre-evaluation phase. The range of valid values is integers between 0-100. A value of 0 means that pre-evaluation never occurs; a value of 100 means pre-evaluation always occurs. The corresponding SAS Management Console option is **Maximum segment ratio**.

MAXSETSIZE
: specifies the maximum number of tuples that can be in a set that combines sets from two different dimensions via a crossjoin or a nonemptycrossjoin. Many times when 2 or more sets are used in a crossjoin, the resulting set is much larger than 1,000,000 tuples. When this option is set to 0, there is no limit to the size of this set.

The default value is 1,000,000 tuples. The corresponding SAS Management Console option is **Maximum number of tuples in a set**.

NECJMULTISETOPT
specifies whether to use the multiset optimization for MDX queries that contain crossjoins on non-empty axes. This optimization reduces the number of subqueries during non-empty crossjoin processing. However, it uses more memory for this processing. In most cases, performance is much better when this optimization is done, because there are fewer subquery calls.

The default value is 1(on). Starting in SAS 9.4, this option will be turned on by default.

NECJMAXMULTISET
specifies the threshold for when the multiset optimization should be used for MDX queries that contain crossjoins on non-empty axes. If the total number of members being crossjoined in an MDX query is greater than the value for *NECJMAXMULTISET*, the non-empty crossjoin multiset optimization does not occur, and the old method of querying the sets is used instead. This option is valid only when the *NECJMULTISETOPT* option is on. When very large sets are crossjoined, the OLAP server can run out of memory. Setting a threshold for this option results in the use of the original method for non-empty crossjoins that requires less memory.

If NECJMAXMULTISET is set to 0 in the OLAPCONFIG file, then no matter how large the set is, the non-empty crossjoin multiset optimization is used. If NECJMAXMULTISET is set to 100000 in the OLAPCONFIG file, then the non-empty crossjoin multiset optimization will only be done if the size of the sets being crossjoined is smaller than 100,000. However, if NECJMAXMULTISET is not set in the OLAPCONFIG file, then a new algorithm is used. The algorithm is based on the set size and the amount of available memory. It determines whether the non-empty crossjoin multiset optimization should be used or not.

NECJMINOPTIMIZE
specifies the threshold for when the non-empty crossjoin optimization should be used for MDX queries that contain crossjoins on non-empty axes. If the total number of members being crossjoined in an MDX query is less than the value for NECJMINOPTIMIZE, the non-empty crossjoin optimization does not occur, and the sets are crossjoined and the empty members removed in later processing. The non-empty crossjoin optimization is done to improve performance of subsequent MDX functions on the set that is being crossjoined. However, if the results set from the crossjoin is small enough, performance will be better if the extra subquery calls required by the non-empty crossjoin optimization are not done.

The default value for this option is 5.

OPTIMIZENECROSSJOIN
specifies whether to optimize MDX queries that contain crossjoins on non-empty axes. The optimizations are done by converting the crossjoin on the non-empty axis to a non-empty crossjoin function. In most cases, performance is much better when this optimization is done, as the set size is reduced for other MDX operations in the query.

Note: Queries are optimized by default.

The default value is 1 (on). The corresponding SAS Management Console option is **Optimize queries that use the NON EMPTY axis clause and the CROSSJOIN function**.

QUERYTO
specifies the amount of time in seconds that a query is allowed to run before timing out. The default value of 0 allows queries to run without any time restriction. If a query runs past the time-out value without completing, then it will terminate with an error message.

The corresponding SAS Management Console option is **MDX Query Timeout (seconds)**.

SPDEMAXTHREADS
specifies the maximum number of threads that can be spawned by the SPD Engine to process individual queries. The range of valid values is 0-8. The default value of 0 causes the SAS OLAP Server to calculate a value between 2 and 8, based on the CPUs that are available on the server.

The corresponding SAS Management Console option is **Number of threads to spawn**.

SQCACHESIZE
specifies the memory size for the subquery cache in MB. If the subquery cache size is so small that the cache does not fit in memory, then part of it is written to disk. This can degrade performance of queries. Also, if the CACHEEMPTYSQ / SQCACHEEMPTY option is turned on so that empty tuples are being cached, then the subquery cache is much larger and might need a larger cache size.

The default value is 5. The corresponding SAS Management Console option is **Subquery cache: Memory Size for Subquery Cache**.

SQCACHEEMPTY
indicates to cache the empty subquery results when this is set to 1. Normally only results that have a value are stored in the subquery cache. When this option is on, all possible subquery results are stored. This requires more memory, especially when the data in the cube is sparse. However, performance can be improved when this option is set because, depending on the query, it could result in fewer subquery calls.

The default value is 0. The corresponding SAS Management Console option is **Subquery cache: Cache the empty subquery results**.

THREADPOOLQRY
specifies the settings for the query thread pool that the SAS OLAP Server uses to process MDX queries. The thread pool is initialized and maintained at a specified minimum number of threads. As the number of queries increases, the number of threads increases towards the specified maximum. If the number of queries decreases, so does the number of threads. If the number of threads reaches the maximum, and if all threads are active, queries are queued and assigned to threads as threads become available. The THREADPOOLQRY option contains the following settings:

MIN
specifies the minimum number of threads. This is the initial number of threads in the query thread pool. The default value is 16.

The corresponding SAS Management Console option is **Minimum number of threads** and is set in the **Advanced Options** for the OLAP server.

Here is an example of the MIN setting:

```
<THREADPOOLQRY MIN="20"/>
```

MAX
specifies the maximum number of threads in the query thread pool. The default value is 64.

The corresponding SAS Management Console option is **Maximum number of threads** and is set in the **Advanced Options** for the OLAP server.

Here is an example of the MAX setting:

```
<THREADPOOLQRY MAX="70"/>
```

MORE
specifies the threshold for adding threads. This is the number of queries queued for assignment to threads as a group. The default value is 0, which means that queries are immediately assigned to threads, as long as threads remain available. Tuning this value optimizes the server's response to rapid increases in query requests.

The corresponding SAS Management Console option is **Threshold for adding threads** and is set in the **Advanced Options** for the OLAP server.

Here is an example of the MORE setting:

```
<THREADPOOLQRY MORE="1"/>
```

TO
specifies the time-out for thread reclamation. This is the time in seconds that a thread can remain inactive before it is removed from the pool. The default value is 30.

The corresponding SAS Management Console option is **Timeout for thread reclamation** and is set in the **Advanced Options** for the OLAP server.

Here is an example of the TO setting:

```
<THREADPOOLQRY TO="35"/>
```

THRSTACKSIZE
specifies the thread stack size. This is the amount of memory in kilobytes that is allocated for each stack in the thread. The default value is 256.

The corresponding SAS Management Console option is **Thread stack size** and is set in the **Advanced Options** for the OLAP server.

Here is an example of the THRSTACKSIZE setting:

```
<THREADPOOLQRY THRSTACKSIZE="300"/>
```

THRFLAGS
specifies the thread options. These are Read-Only binary thread options. No options are currently defined.

The corresponding SAS Management Console option is **Thread options**.

WORKPATH
specifies the path for temporary work files used by the SPD Engine. The default is the working directory of the SAS OLAP server.

The corresponding SAS Management Console option is **Path for temporary working files**.

For further information, see the topic “Tuning SAS OLAP Servers with Advanced Server Options” in the *SAS Intelligence Platform: Application Server Administration Guide*.

SAS Management Console

SAS Management Console provides a single point of access and control to perform the administrative tasks required to create and maintain an integrated SAS environment across multiple platforms. SAS Management Console uses a plug-in architecture that is specific to your SAS installation and environment. With SAS Management Console, you can perform the following tasks:

- manage the metadata for your SAS data and environment
- maintain library and server definitions
- create, manage and maintain authorization information for your users and groups
- administer resource access control templates

Note: For further information about SAS Management Console see the Help in SAS Management Console.

SAS OLAP Server Monitor

The SAS OLAP Server Monitor plug-in for SAS Management Console enables you to manage tasks that are specific to the SAS OLAP Server. You can configure SAS OLAP Servers, administer the connection to a SAS OLAP Server, monitor sessions that are running on a SAS OLAP Server, and monitor queries that are being processed by a SAS OLAP Server. You can also enable and disable cubes and refresh cube metadata. For further information about SAS OLAP Server Monitor, see the Help for SAS OLAP Server Monitor in SAS Management Console.

SAS OLAP Cube Studio

SAS OLAP Cube Studio provides cube designers with an easy-to-use graphical user interface for creating SAS OLAP cubes. You can build SAS OLAP cubes, edit cubes, incrementally update cubes, tune aggregations, and make various other modifications to existing cubes. The SAS OLAP Server: User's Guide covers the different functions and features available for creating SAS OLAP cubes. See “ SAS OLAP Cube Studio and the OLAP Procedure” on page 60 for more information.

OLAP Schema

A SAS OLAP schema is an organization container for SAS OLAP cubes. A SAS OLAP schema specifies the location of a set of cubes. It communicates with a SAS OLAP Server about which cubes can be accessed by the server and then queried. When you install your SAS OLAP Server, a default SAS OLAP schema is created named SASMain – OLAP Schema. If needed, you can have multiple schemas. However, you can have only one schema assigned to an OLAP server at a given time. For more information about SAS OLAP schemas, see “OLAP Schemas” on page 30. Also, see “Managing OLAP Cube Data” in the *SAS Intelligence Platform: Data Administration Guide*.

SAS Libraries

A SAS library is a collection of one or more files that are referenced and stored as a unit. For SAS OLAP, libraries serve as organization containers for the data tables that you use to create SAS OLAP cubes. You can have numerous libraries to organize your data tables with. To define your libraries, you can use the New Library wizard in SAS OLAP Cube Studio or in SAS Management Console. For more information about defining and managing libraries, see “SAS Libraries and Tables” on page 33 and “Assigning Libraries” in the *SAS Intelligence Platform: Application Server Administration Guide*.

Data Tables

Data tables are frequently used to define data stores. They can be used to define data stores, summary data, a join, or a table that holds information that does not conform to any other data storage type. A detail, or base, table is any data table defined in the SAS metadata that contains the measures and levels for a cube. The detail table consists of unsummarized data that must include one column for each level and one numeric analysis column for each set of measures that will be generated. There are different types of data tables that can be used to build a SAS OLAP cube. You can use the following tables to construct SAS OLAP cubes:

- detail tables (base tables)
- star schemas (a fact table and associated dimension tables)
- summary tables (NWAY data set)
- drill-through tables
- aggregation tables

For detailed information about the different data tables, see “Data Tables Used to Define SAS OLAP Cubes” on page 42 . Also, see the *SAS Intelligence Platform: Data Administration Guide* for more information about managing data table sources.

Authorization Permissions

When creating and editing SAS OLAP cubes, you must establish and apply security permissions to ensure the correct access to cube data among the different users on a system. SAS authorization permissions can be set at general levels or fine-grain levels and are conveyed across user identities (user, group) and objects (folder, table, library). Authorization permission settings are then inherited from parent identities or objects.

SAS OLAP cubes and the different components of a cube (dimensions, hierarchies, levels, measures, calculated measures, and cube jobs) can have authorization permissions set using the authorization functions that are available in SAS OLAP Cube Studio and in the Authorization Manager plug-in in SAS Management Console. For further information about establishing security and authorization permissions for your SAS OLAP environment, see “Securing Cubes” on page 119 . In addition, see the *SAS Intelligence Platform: Security Administration Guide* for detailed information about security administration for your SAS environment.

LOCKDOWN

There may be situations where you want to control access to certain data or files in your SAS environment. You can limit the reach and activities of a SAS server by putting it in a locked-down state. The lockdown feature limits availability to all resources in your SAS Environment except those that are defined in a list. The lockdown feature gives SAS processes access to specific, identified physical resources on the server.

When a server enters the lockdown state, the server can access the following resouces:

- libraries and files that are defined in metadata or in autoexec files or through INITSTMT execution.
- host paths and files that are included in the server’s list of permitted resources, known as the lockdown path list. The lockdown path list is established during server initialization and is finalized at the lockdown point. The lockdown path list is also referred to as a whitelist that specifies which paths are accessible.

If lockdown is being implemented, SAS enters the lockdown state after initialization of the SAS environment. For SAS OLAP Server, you must use the LOCKDOWN system option in addition to the LOCKDOWN statement.

The LOCKDOWN system option enables limited access to files and to specific SAS features for a SAS session executing in a batch or server processing mode. The default setting is NOLOCKDOWN. The LOCKDOWN system option is added to the sasv9_usermods.cfg file that is located in the server’s configuration directory. You

can use the LOCKDOWN system option in a configuration file or during a SAS invocation.

Here is the syntax for use with the LOCKDOWN option:

LOCKDOWN | NOLOCKDOWN

Here is an example of the LOCKDOWN system option:

```
/usr/local/bin/SAS/SASFoundation/9.4/sas -objectserver
-lockdown -autoexec sample-auto.sas
```

The LOCKDOWN statement is added to the autoexec_usermods.sas file that is located in the server's configuration directory. Use the statement's PATH= argument to specify which directories and files are available to the server's users. SAS automatically adds subdirectories of any valid directories in the list to the lockdown path list. A pathname can be a relative pathname or an absolute pathname.

Here is the syntax for use with the LOCKDOWN statement:

LOCKDOWN PATH= 'pathname-1' <'pathname-n'…>;

Here is an example of a LOCKDOWN statement for SAS application server control:

```
lockdown path="~" "/SASData/reportdata" "/SASData/STPData" "/SASData/
sales";
```

For further information about lockdown, see the following topics:

- "Locked-Down Servers" in the *SAS Intelligence Platform: Security Administration Guide*
- LOCKDOWN Statement in the *SAS Intelligence Platform: Application Server Administration Guide*
- LOCKDOWN System Option in the *SAS Intelligence Platform: Application Server Administration Guide*

4

Organization and Management of Your Data

Cube Data Organization

When working with SAS OLAP Cube data, you must manage the storage and organization of your cube input tables and the cube data, including the cube metadata and the corresponding physical cube files. During the course of developing and executing SAS OLAP Cubes, you can do the following:

- define the cube source data and libraries
- establish the OLAP schema assignment for your cubes
- create folders in which to store your cubes
- manage any cube jobs that you deploy

OLAP Schemas

Overview

A SAS OLAP schema represents a group of cubes in the SAS metadata and specifies which group of cubes an OLAP server can access. An initial OLAP schema is assigned to a SAS OLAP Server when that server is defined in the metadata. Although you can have multiple schemas, a server can access only the cubes in one schema at a time. As a result, you do not need to create more OLAP schemas than there are OLAP servers on your system.

Once an OLAP schema is available, a cube can be assigned to that OLAP schema during the cube building process. Whereas there are no absolute restrictions for the number of cubes per OLAP schema, assigning a large number of cubes to a schema should be avoided. This is because the cubes compete for the OLAP server's cube cache and data cache at the time they are accessed and handled by the server.

You can create and maintain OLAP schemas in SAS OLAP Cube Studio or SAS Management Console. Here are some of the tasks that you can perform:

Defining an OLAP Schema

In addition to SAS Workspace Server and SAS OLAP Server definitions, you must also have an OLAP schema defined in the active metadata server. The standard SAS deployment process creates an OLAP schema named SASApp - OLAP Schema and assigns it to the SAS OLAP Server that is also configured during

deployment. To define new schemas, you can use the New OLAP Schema Wizard, which is available from SAS OLAP Cube Studio and SAS Management Console. In SAS Management Console, you can launch the OLAP Schema Wizard when you view the **OLAP Schema** tab on the Properties dialog box for an existing server. These steps explain how to use SAS OLAP Cube Studio to launch the OLAP Schema Wizard and define a new schema:

1 Connect to the SAS Metadata Server.

2 Select **File** ⇨ **New** ⇨ **OLAP Schema**.

3 On the General page of the New OLAP Schema Wizard, enter the schema name and description. Click **Next**.

4 On the Server Assignment page, specify the OLAP servers that can access the schema. This step is optional. Click **Next**.

Note: If you choose not to specify the server using the OLAP Schema Wizard, you can add that information later by modifying the schema's properties.

5 On the Finish page, click **Finish**.

Note: When defining a new SAS OLAP Server, if you accept the default definition settings, then an OLAP schema is automatically created and assigned to the server. To change that assignment to a different OLAP schema, you must edit the server definition. In SAS Management Console, select the Server Manager plug-in to make these changes.

Editing an OLAP Schema

The OLAP schema specifies which group of cubes an OLAP server can access. To edit an OLAP schema in SAS OLAP Cube Studio, complete these steps:

1 In the Tree View, select the OLAP schema that you want to edit. Then select **Properties** from the **Edit** menu or from the schema's context menu.

2 In the Properties dialog box, make changes on the **General** and **Server Assignment** tabs.

You can view cubes assigned to the schema on the **Cubes** tab. To change the OLAP schema assignment for a cube, you must select the properties for a cube and select the **Location** tab. From here you can enter the name of a different OLAP schema.

3 When you are finished, click **OK** on the Properties dialog box.

Changing the OLAP Schema for a Cube

When working with SAS OLAP Cubes, it might be necessary to assign a cube to a different schema. You can change the OLAP schema assignment for a cube by selecting **Change OLAP Schema** from the **Actions** menu or the cube's context menu. The Change OLAP Schema dialog box appears. The selected cube is currently assigned to the OLAP schema that is listed in the dialog box.

Select an OLAP schema from the drop-down list of OLAP schemas. If needed, you can also create a new OLAP schema by selecting the **New** button. This opens the New SAS OLAP Schema wizard. After you have selected an OLAP schema, select the **OK** button. The cube is now assigned to the selected schema.

Note: When reassigning a SAS OLAP cube to a different OLAP schema, be careful to note which OLAP server the schema is assigned to.

Deleting OLAP Schemas

To delete an OLAP schema, complete the following steps:

1. In the Tree View, select the schema that you want to delete.
2. Select **Delete** from the **Edit** menu or from the schema's context menu.
3. Click **Yes** on the Confirm Delete dialog box.

Note: You cannot delete a schema that currently has OLAP cubes assigned to it. In addition, you must have the necessary authorizations.

Viewing the OLAP Schema Properties

In SAS OLAP Cube Studio, you can view the properties for a SAS OLAP schema by selecting the schema from the Tree View and then selecting the **Properties** function. The Properties function can be accessed from the **Edit** menu and from the schema's context menu. The OLAP Schema Properties dialog box contains tabs that provide general information as well as information that is specific to SAS OLAP schemas. The following tabs are specific to OLAP schemas:

Server Assignment
lists the OLAP servers defined in the current metadata. On the **Server Assignment** tab, you can select an OLAP server from the **Available** list and move it to the **Selected** list.

Cubes
lists cubes that are assigned to the current schema. The **Cubes** tab also displays whether the cubes physically exist or exist as metadata only.

Assigning the OLAP Schema with the OLAP Procedure

To assign an OLAP schema, you must use the OLAP_SCHEMA= option with the METASVR statement. Here is an example of the METASVR statement and the OLAP_SCHEMA= option:

```
METASVR
   HOST  = "hw4195.ec.sas.com"
   PORT   = 8561
  OLAP_SCHEMA = "SASApp - OLAP Schema";
```

Note: For more information about SAS OLAP schemas see “Managing OLAP Cube Data” in the *SAS Intelligence Platform: Data Administration Guide*.

SAS Libraries and Tables

Defining Libraries and Tables

When you set up the SAS OLAP Cube Studio environment, you must define the data tables that are used to build cubes and the libraries that they are assigned to. The following tasks must be completed:

- A metadata server must be started.
- A workspace server must be started and registered in the metadata.
- The metadata for the data tables and libraries must be registered in the metadata and stored on the workspace server.

Note: You must have ReadMetadata and WriteMetadata permissions to perform these tasks.

The tables and libraries can be defined in SAS applications such as SAS Data Integration Studio or SAS Management Console before you set up your SAS OLAP Cube Studio environment. If however, this is not done beforehand, you can define them in SAS OLAP Cube Studio. You need to have your tables and libraries defined if you plan to do either of these tasks:

- create the physical cube in addition to registering its metadata
- manually add, modify, or drop specific aggregations for a cube

You can use the Source Designer wizard to define your data tables. Use the New Library wizard to define your libraries.

Creating a New Library Definition for Source Data Tables

You can create a new library definition for your source data tables after you start a SAS Metadata Server and define a SAS Workspace Server in a SAS metadata folder. You can create new library definitions using the New Library Wizard, which is available from SAS OLAP Cube Studio and SAS Management Console.

If you have more than one SAS Workspace Server defined, you should assign the library to all of the workspace servers that might be used to create cubes. Otherwise, SAS OLAP Cube Studio attempts to download the data to the server where the cube is being built. This requires SAS/CONNECT and might not be the most efficient way to build a cube.

These steps explain how to use SAS OLAP Cube Studio to launch the New Library Wizard and define a new library:

1. Connect to the SAS Metadata Server.
2. Select **File** ⇨ **New** ⇨ **Library**. This opens the New Library Wizard.
3. Select a Resource Template and then click **Next** to continue.

   Note: If you connected to the SAS Metadata Server with an unrestricted metadata profile, you are prompted to enter your user ID and password.

4. In the New Library Wizard, enter the library name. You can also enter an optional description. Click **Next**.
5. Enter values for the libref, the Engine (BASE is the default), the Content Server, and the Path Specification fields. Follow these guidelines:
   - The libref is a short name (or alias) for the full physical name of a SAS library (for example, sasuser).
   - The path specifies the physical location of the tables contained in the library that you are defining. Select an existing path from the box or click **New** to enter a new path.
   - If your data is accessed through a WebDAV content server, select the **Enable webDAV Support** check box. The content server specifies the HTTP server that is used to access the data.
   - Click the **Advanced Options** button to set host-independent options such as file encoding, as well as host-specific options. Click **OK** to close the dialog box and return to the New Library Wizard.
   - Click **Next**.
6. Select the SAS Workspace Servers on which the new library is to reside. Click **Next**.
7. Click **Finish** to complete the new library definition.

Note: For further information about SAS Libraries see “Assigning Libraries” in the *SAS Intelligence Platform: Data Administration Guide*.

Defining Tables Used to Build Cubes

You define tables using the Source Designer wizard, which is available from SAS OLAP Cube Studio and SAS Data Integration Studio. Here is a list of the tables that can be used to define a cube:

- detail tables (unsummarized data)
- fact tables and dimension tables (for cubes based on star schemas)
- aggregation tables (fully summarized external tables)
- drill-through tables (views maintained by the user that represent all of the data used to define a cube)

These steps explain how to use SAS OLAP Cube Studio to launch the Source Designer wizard and define new tables:

1. Connect to the SAS Metadata Server.
2. Select **File** ⇨ **Register Table**. This opens the Source Designer wizard. The Source Designer wizard is also available from the Cube Designer wizard.
3. Select the SAS data source and click **Next**.
4. Select the name of the SAS library that points to the tables that you are importing from the current SAS Workspace Server. Click **Next** to see a list of SAS data sets in the selected library. (You can also click **New** to create a new library.)
5. Select the data sets that you want to load into the metadata, and click **Next**.
6. Click **Finish**.

Note: If you previously selected a server context and tested the SAS Workspace Server connection, you might not be prompted for the name of the server context.

Renaming Tables

Introduction

In SAS OLAP Cube Studio, you can change the name of a data table that you have registered in the SAS metadata folder. The table name that you see in the Tree View is the display name for the table, similar to a label or short description. You must have WriteMetadata permission on the table for which you want to view properties.

Renaming a Table from the Tree View

To rename a table from the Tree View, complete the following steps.

1. In the Tree View, select the **Inventory** tab. Select a data table from the **Table** node.
2. Select the **Rename** function from the **Edit** menu or from the table's context menu. The table name in the Tree View is highlighted.
3. Enter a new name and select **ENTER**.

Renaming a Table from the Properties Dialog Box

To rename a table from the Properties dialog box, complete the following steps.

1. In the Tree View, select the **Inventory** tab. Select a data table from the **Table** node.
2. Select the **Properties** function from the **Edit** menu or from the table's context menu.
3. Select the **General** tab.
4. Enter a new name in the **Name** field.
5. Click **OK** to save the change and close the dialog box, or click **Apply** to save the change and remain in the dialog box.

For information about naming tables, see “Naming Guidelines and Rules for the SAS OLAP Server ” on page 51 .

Deleting Tables

You can delete tables that you have registered in the SAS metadata. In order to delete a table, you must have been granted WriteMetadata permission on that table. Complete the following steps to delete a table.

1. In the Tree View, select the **Inventory** tab.
2. Select **Tables** and open the list of tables.
3. Select the table that you want to delete.
4. Select **Delete** from the **Edit** menu or from the table's context menu.
5. Click **Yes** in the Confirm Delete message box. The table is deleted.

Viewing Table Data in SAS OLAP Cube Studio

In SAS OLAP Cube Studio, you can view the underlying data of the tables that are available to build SAS OLAP cubes with. Tables are listed under the **Tables** node on the **Inventory** tab of the Tree View. You can view the underlying data for a table by selecting **View Data** from the **Actions** menu or from the table's context menu.

Note: For further information about managing tables see " Managing Table Metadata" in the *SAS Intelligence Platform: Data Administration Guide*.

Managing Folders in SAS OLAP Cube Studio

SAS Metadata Folders

In order to define cubes, you must be connected to a running metadata server and have a SAS folder to store your cubes to. A SAS folder is a location in which data, metadata, or programs are stored, organized, and maintained. SAS folders are accessible to users either directly or through a network. You can create and edit SAS folders in SAS OLAP Cube Studio, SAS Management Console, or SAS Data Integration Studio.

To create a new folder in SAS OLAP Cube Studio, select **File** ⇨ **New** ⇨ **Folder**. The New Folder dialog box appears. Enter the name and location of the folder and select **OK**.

Finding an Object in a SAS Folder

When you are working in the **Inventory** tab of the Tree View, you can quickly locate which folder an object is stored in with the Find In Folders function. Select an object in the **Inventory** tab of the Tree View. Select **Find In Folders** from the **Edit** menu or from the object's context menu. The Tree View then displays the folder location of the object.

Copying an Object to a Different SAS Folder

In SAS OLAP Cube Studio, you can copy a cube object from one folder to another.

1. Select an object in the Tree View.
2. Select **Copy to Folder** from the **Edit** menu or from the object's context menu. The Select a Location dialog box appears.
3. Select a folder from the folders list and click **OK**. The object is copied to the new folder.

Note: You must have WriteMemberMetadata (WMM) permission on the folder that you are copying objects to. For further information about permissions for different objects, see the *SAS Intelligence Platform: Security Administration Guide*.

Moving Objects between SAS Folders

In SAS OLAP Cube Studio, you can move an object from one folder to another.

1. Select an object in the Tree View.
2. Select **Move to Folder** from the **Edit** menu or from the object's context menu. The Select a Location dialog box appears.
3. Select a folder from the folders list and click **OK**. The object is moved to the new folder.

Note: You must have WMM permission on the folder that you are moving objects to. For further information about permissions for different objects, see the *SAS Intelligence Platform: Security Administration Guide*.

Further Information

For more information about working with SAS folders, see “Working with SAS Folders” in the *SAS Intelligence Platform: System Administration Guide*.

SAS OLAP Cube Jobs

SAS OLAP Cube Jobs

When you create a cube definition in SAS OLAP Cube Studio, a cube job is automatically created for that cube. A job is a collection of SAS tasks that create output. The cube job is the metadata that links a cube to its load data table. This can be a detail table, a fact table, or a fully summarized table. The cube job also contains a classifier map in the metadata which links the columns in the load table to the levels, measures, and member properties for the cube.

Note: The cube job does not link the star schema dimension tables or the aggregation tables to the cube.

After a cube definition and the cube job is created, you can perform the following tasks:

- deploy and redeploy a cube job for scheduling
- export a SAS package
- view the properties for a cube job

When you deploy a cube job, the code needed to create the cube is generated. It is then stored in a file in the specified deployment directory. In addition, a deployed job object is created. This new object is linked to the original cube job and it can be added to a SAS flow for scheduling. You can also now redeploy the job as needed.

Cube Job Deployment and Redeployment

SAS OLAP cube jobs are deployed to create cubes and are listed in the `Job (cube)` folder under the **Inventory** tab of the Tree View. To deploy a cube job, select the job in the Tree View and select **Actions** ⇨ **Deploy**. The Deploy a job for scheduling dialog box appears. From here you can enter information about the job. If the job is deployed successfully, you receive a confirmation message.

The deployed job is then listed in the `Deployed job` folder in the Tree View. Each time you deploy a job, a separate deployed job object is created and listed in the Deployed job folder. For example: If you deploy a SAS cube job based on the cube "Sales", the deployed job is labeled as "Sales". If you deploy that same cube job again, the next deployed job is labeled "Sales000". Deploy the cube job a third time and it is labeled "Sales001".

Note: Deployed jobs for a cube job can also be accessed from that job's context menu. Right-click on the cube job and select **Scheduling**. The deployed jobs are listed in the **Scheduling** drop-down list. A deployed cube job can also be

redeployed after the initial deployment. Select a deployed cube job from the `Deployed job` folder in the Tree View. Then, select **Actions** ⇨ **Redeploy**.

Job and Deployed Job Properties

Job Properties

In SAS OLAP Cube Studio, you can view the properties for an OLAP cube job or a deployed job. Select a job from the Tree View and select **Properties** from the **Edit** menu or from the job's context menu. The Properties dialog box then opens. A cube job has the same standard dialog box tabs as other cube objects. In addition, it also contains the following tabs:

Process
The **Process** tab contains options to specify how the job's code is generated (Code Generation).

Pre and Post Process
The **Pre and Post Process** tab contains options to insert user-written code at the beginning or end of the current job.

Deployed Job Properties

A deployed job has the same standard dialog box tabs as other cube objects. It also contains the **Scheduling Details** tab for the deployment. This tab includes the command that is used to execute the current job in batch mode. It also includes the settings for the SAS Application Server that is used to deploy the job.

5

Planning for SAS OLAP Cubes

Overview

The goal of an OLAP system is to have data that is organized, available, and presented as relevant information to decision makers. To successfully build and query SAS OLAP cubes, the data in your input files must be internally consistent. In addition, the columns of input data sets need to reflect the dimensional levels of the cubes. When building an OLAP application, it is beneficial to assume that building the actual cube is the smaller part of the project. The larger task includes finding and collecting the data and making that data consistent.

When planning for a SAS OLAP cube, you should consider the following items:

- the data tables that you use to build the cube
- aggregation design
- aggregation storage considerations (MOLAP or ROLAP)
- size specifications for SAS OLAP cubes and cube objects
- naming guidelines and rules for SAS OLAP cubes and cube objects
- SAS formats that are available for cube measures
- statistics that are available for cube measures

Data Tables Used to Define SAS OLAP Cubes

Overview

A cube is always loaded from data in relational tables. This data can be stored either in SAS tables or in external RDBMSs and can be accessed through a wide selection of SAS data engines, including the Base SAS engine, SPD Server, SPD Engine, and the SAS/ACCESS engines to external RDBMSs. SAS software enables you to be independent of the physical storage format of the data. When building a SAS OLAP cube, you must consider the format that your source data is stored in and how that source data is to be used in the SAS OLAP cube. Cubes can be loaded from data that is contained in any of the following types of tables:

- detail tables
- fact tables and dimension tables (star schema)
- summary tables
- aggregation tables

- drill-through tables

Detail Tables

A detail, or base, table is any table defined in the SAS metadata that contains the measures and levels for a cube. A detail table consists of unsummarized data that must include one column for each level and one numeric analysis column for each set of measures that will be generated. A detail table includes all the columns that are needed to load the hierarchy levels, the level properties, and measures. Specifically, a detail table must contain the following elements:

- one column per dimension level (character or numeric)
- one column for each property (character or numeric)
- numeric analysis variables

Fact Tables and Dimension Tables (Star Schema)

A star schema consists of a single fact table and one or more dimension tables. The fact table must contain one numeric analysis column for each set of measures that will be generated. You cannot have more than one property value for each distinct value of the corresponding level. To successfully load a cube, all foreign keys in the fact table need to have a corresponding primary key in a dimension table. A star schema configuration is organized with the following requirements and considerations:

- The input columns for data measures must be stored in the fact table.
- The input columns for dimension levels and properties must be stored in one table for each dimension.
- The data records of the fact table and the dimension tables must be linked via primary and foreign keys.
- If the dimension levels are defined in a dimension table, all the level columns for that dimension must be contained in the same dimension table.
- Both the dimension keys and fact keys are single columns, not combinations of columns.
- A dimension can be in the fact table. In this case, all the level columns are in the fact table and no fact or dimension key is required.
- The dimension key can also be a level in the dimension.

The fact table requires the following conditions:

- one key column with a foreign key for each dimension (character or numeric)
- numeric analysis variables

A dimension table requires the following conditions:

- one key column with a primary key (character or numeric)
- one column for each dimension level (character or numeric)

- one column for each property (character or numeric)

Summary Tables

The summary table is also known as the NWAY data set. It is a table that is already summarized and requires the following elements:

- one column per dimension level (character or numeric)
- one column for each property (character or numeric)
- one column per stored measure, summarized by the appropriate statistic for the measure

Aggregation Tables

In addition to a summary table for the NWAY aggregation, summary tables can also be provided for other aggregations. Aggregation tables contain presummarized data for any combination of dimension levels. This feature enables cubes to access summarized numbers quickly. Aggregated data can be created and automatically stored in MOLAP (Multidimensional OLAP) tables as the cube is built or manually stored in ROLAP (Relational OLAP) tables and linked into the cube.

All aggregation tables must contain a column for each measure in the cube where the statistic for the measure is one of the following: N, NMISS, SUM, MAX, MIN, or USS. An aggregation table can be used in two ways:

- as an NWAY data source for the cube. In this case, the table must contain a column for every level in the cube and a column for every stored measure.
- as a subaggregation for the cube. In this case, the table must include a column for each level of the aggregation and a column for every stored measure.

When planning your aggregation tables, note that the column names and attributes (type, format) must be consistent across all input tables for a cube. Aggregation tables require the following structure and organization.

- one column for each dimension level in the aggregation
- one column per measure

Drill-Through Tables

Many SAS OLAP applications give users the ability to select a cell or a range of cells and then view the input data that the cell data was summarized from. Drill-through tables can be used by client applications to provide a view from processed data into the underlying data source. Drill-through tables are views that represent all of the input data used to define a cube. Assigning a drill-through table to a cube is optional.

When you are evaluating the data source for a drill-through table, you should consider the following requirements:

- The drill-through table must have the same columns with the same attributes (name, type, format) as the tables that the cube was loaded from.
- In many cases, the detail table is also the drill-through table.
- For a star schema, a drill-through table is created with a view that fully joins the star schema. For a summary table, that summary table can be used as a drill-through table.

Note: When you select a data table for drill-through, you might need to consider user access and security restrictions for that table. For further information, see “Security for Drill-through Tables” on page 126 “Security for Drill-through Tables” on page 126 .

Aggregation Design

Overview

Efficient drilling or traversing of the cube data is a key factor in flexible and swift decision making and analysis. In order to maintain speed and consistency in reporting, data is usually pre-calculated or aggregated. An important factor in query performance is good aggregation design, which includes decisions about total storage space, available build time, storage location, and storage format.

Aggregation Size

When planning your data storage and design, it is helpful to approximate the size of aggregations. A basis for estimating aggregation size is the number of distinct values in a dimension level, otherwise known as cardinality. The other factor that determines aggregations size is density. Density is a measure of how many members of each dimension in an aggregation occur in combination with the members of the other dimensions (For example, there might not be sales of a specific product on a specific date). The total cube size, as well as the resources that are available for the cube build process, determine the build time that is needed. It is also important to note that build time should not exceed the cube update interval.

Aggregation size and available hardware influence your choices for aggregation partitioning. You can separate aggregations into multiple files. A reduced file size might accelerate OLAP server access time, particularly if multiple processors are available for multi-threaded processing. You can use either pre-aggregated summary tables, the cube's own efficient aggregation storage, or a combination of

both. Using indexes on either storage type might increase query performance, while also increasing storage space and build time.

User Query Patterns and ARM Logging

Overview

In choosing the best aggregations to summarize and store with the cube, the most important factor to take into account is user query behavior. It is recommended that you start with an initial aggregation design that is based on a minimal set of aggregations or on your best assumptions about usage patterns.

After the cube is deployed to users, you can use the ARM (Application Response Management) logging capabilities of the SAS OLAP Server to collect data about the usage pattern and the performance of individual queries. You can analyze the collected data to find out which cube aggregations are used most, which aggregations you can safely eliminate without harming query performance, and which aggregations are often requested but don't exist in the cube. The ARM data provides all the information that you need in order to get the optimal query improvement for your build time and cube storage space.

Administering ARM Logs

When using ARM logs to generate aggregations for a cube, you must consider the administration and maintenance of the ARM logs. If you are using multiple load-balanced servers, you must plan for the naming of the ARM logs that these servers write to. Each load-balanced server that you use writes to a designated ARM log file. If the ARM log filename is the same for the different load-balanced servers, then the servers will write to the same ARM log simultaneously. This can produce an invalid ARM log. It is recommended that you provide a unique ARM log name for each load-balanced server to write ARM log data to. You can then select each ARM log file to analyze in the Aggregation Tuning dialog box.

Note: For further information about ARM logging, see the topic “ARM Logging” in the chapter “Administering SAS OLAP Servers” in the *SAS Intelligence Platform: Application Server Administration Guide* and the *SAS Application Response Measurement (ARM) Reference*.

Aggregation Performance Settings

When planning and creating aggregations for a cube, you can customize various performance settings for a single aggregation or for all aggregations in the cube. You can change the following aggregation-specific settings:

- ASYNCINDEXLIMIT=
- COMPACT_NWAY
- COMPRESS | NOCOMPRESS
- CONCURRENT=
- DATAPATH=
- INDEXPATH=
- INDEXSORTSIZE=
- MAXTHREADS=
- INDEX|NOINDEX
- PARTSIZE=
- SEGSIZE=

Note: ASYNCINDEXLIMIT=, CONCURRENT=, INDEXSORTSIZE=, and MAXTHREADS= are available only in the PROC OLAP statement.

You can change these settings using either the OLAP procedure or in SAS OLAP Cube Studio. In SAS OLAP Cube Studio, you can apply these settings from these functions:

- Cube Designer – Aggregations dialog box
- Aggregation Tuning dialog box

Aggregation Storage

Overview

When determining how to efficiently deliver data in a multidimensional cube, how data is stored is an important factor. Summarizing data with aggregations improves query response. Any combination of dimension levels can become a stored aggregation. Which aggregations are being stored has a direct effect on the SAS OLAP Server CPU usage, file I/O, and query response times. The aggregations that are being stored also affect cube build time and the absolute cube file size. Therefore, it is a trade-off between a single instance of resource use at cube build time and multiple instances of resource use at cube query time.

The aggregated data values for SAS OLAP cubes can be stored either with the cube in the cube's internal format or external to the cube in relational summary tables.

MOLAP
: MOLAP aggregation storage is the cube-internal storage for aggregations. MOLAP aggregation tables are created as part of the cube creation step.

ROLAP
: ROLAP aggregation storage is the cube-external summary tables. ROLAP summary tables need to be precalculated from the input data (using tools such as SQL or PROC MEANS/PROC SUMMARY) and made known to the cube at cube creation time.

MOLAP Aggregation Storage

SAS MOLAP aggregation storage maintains the cube data in the same table format as the format that is used by the SAS Scalable Performance Data (SPD) Engine. MOLAP aggregation storage takes advantage of key contraction and allows data access by using the cube's internal data representation directly.

MOLAP aggregation storage of SAS OLAP cubes has the same threading and scalability features as the files used by the SAS SPD Engine. The data and the index section of the files are stored in different physical files. This enables parallel access to the data and index sections. The data and index files themselves are stored in partitions, enabling parallel data retrieval within the same file. The partitions of the data and the index section can be distributed over multiple disc controllers, thus adding a further boost to the threaded and partitioned architecture by reducing contention and possible bottlenecks in the physical I/O.

ROLAP Aggregation Storage

ROLAP tables used in SAS OLAP cubes can be SAS data sets, SAS data views, or any tables or views accessible through a SAS engine. This extends the choice of available storage options for SAS OLAP cubes to include SPDE, SPDS, and any RDBMS product for which a SAS/ACCESS software product is available.

ROLAP aggregation tables must conform to the structure of the input data. The columns that feed the dimension levels must have the same column names and attributes that were used in the input data when loading the cube. In addition, all aggregations must be stored in fully de-normalized form. Here are some guidelines to make aggregation columns for measures available:

- Each ROLAP aggregation table must include all columns for the cube's measures with stored statistics.
- SAS OLAP cube aggregations store the following statistics: SUM, N, NMISS, USS, MIN, and MAX. Other available statistics are derived from the stored statistics by internal calculations. For example, in order to include a measure for the AVG statistic in your cube, you need to make columns available in your ROLAP aggregation tables that were generated by using SUM and N (count).

ROLAP data is stored in either a flat file or with a star schema. With ROLAP, each instance of slicing and dicing of data is part of an SQL query (or multiple SQL queries) and is comparable to a WHERE clause in the SQL statement.

ROLAP data requests can also run against the data that was used to create and load the cube, whether from a detail table or a star schema. The cube's input data can be used in place of the aggregation with the combination of the lowest level of all dimensions (often called NWAY or NWAY aggregation, which is a name borrowed

from PROC MEANS/PROC SUMMARY where it denotes the combination of all CLASS variables).

Choosing MOLAP or ROLAP Aggregation Storage

MOLAP aggregation storage is optimized for SAS OLAP Server internal processing and has a minimal data-size footprint. It uses threaded, parallel data access and is tunable to any hardware environment. MOLAP aggregation storage is convenient because it doesn't require additional data management steps.

ROLAP aggregation storage enables you to use existing ROLAP schemas and reuse legacy SAS OLAP Server, SAS 8 HOLAP structures. ROLAP aggregation storage enables users to use database systems to store and serve cube aggregation data and to off-load data access, I/O, and roll-up.

A hybrid approach is possible. For example, users with existing ROLAP structures can build a “light” SAS OLAP cube with no additional stored aggregations and add MOLAP aggregations to further tune the cube performance. This is a useful technique for OLAP users who have moved from MOLAP to a ROLAP cube structure to utilize DBMS data directly. Sometimes, the on-the-fly query performance of SQL processing for ROLAP-based MDX queries does not match closely enough to the speed of pre-summarized MOLAP performance. For this reason, the SAS OLAP Server allows users to add aggregations to their ROLAP cube. This step changes the approach from a purely ROLAP structure to a hybrid approach. It’s called hybrid because it is combining the benefits of pre-summarization in MOLAP with the benefits of ROLAP. By adding aggregations for the most commonly used data to your ROLAP cube, query performance can be enhanced. This hybrid OLAP approach supports a shorter cube build time, compared to MOLAP, and can improve ROLAP query performance. This is what is referred to by the term HOLAP.

Improving ROLAP Throughput Performance with SPDS

Beginning in SAS 9.4, you can enable execution of user-defined (FORMAT procedure style) formats in SPDS and enable the OLAP server to use them when querying formatted values with the PUT function. SPDS is a storage solution based on SAS that is able to handle the processing of formatted computations in parallel. The SAS PUT function has been added to the SPDS LIBNAME function table. This allows the PUT function to pass in the SQL query to SPDS. SPDS can handle formats with PUT, both intrinsic to SAS and user-defined (Proc Format style). This capability can increase performance for the OLAP server when used with SPDS. When the PUT function is used, MDX query response times against ROLAP cubes will improve.

In support of the PUT function, the option SQL_IP_TRACE=SOURCE has been added. When used with option MSGLEVEL=I, it turns on additional tracing and messaging to the SAS log. These messages allow you to inspect whether the PUT function is passed to SPDS. The functionality is enabled by setting the following options:

1 IP=YES is set for the SPDS LIBNAME (it is set to NO by default).

2 The FMTSEARCH option is set to the "formats" domain in the SPDS LIBNAME. This enables Proc SQL to utilize the SPDS format.catalog during its parse and validate phases.

For more information, See “User Defined Formats” in the *SAS Scalable Performance Data Server: User’s Guide*.

SAS OLAP Cube Size Specifications

Overview

When you are creating SAS OLAP cubes, there are some size specifications for the various components of the cube.

Defining the Number of Dimensions for a Cube

The maximum number of dimensions that can be defined in a cube is determined by combining the number of dimensions with the number of multiple hierarchies. The maximum value of that sum is 128. Mathematically, the sum is expressed as follows:

MaxDims = NumDims + NumMultipleHeirarchies = 128

All hierarchies other than the first hierarchy in each dimension apply to the total. The following are some examples of cubes that meet the maximum number of dimensions:

- 128 dimensions, each dimension has 1 hierarchy
- 127 dimensions, 1 dimension has 2 hierarchies
- 126 dimensions, 1 dimension has 3 hierarchies
- 126 dimensions, 2 dimensions have 2 hierarchies

Defining the Number of Hierarchies for a Cube

When specifying hierarchies, you must define at least one hierarchy for each dimension. The maximum number of hierarchies that can be defined in a cube is determined by combining the number of multiple hierarchies with the number of dimensions. The maximum value of that sum is 128. Mathematically, the sum is expressed as follows:

MaxHeirs = NumMultHiers + NumDimensions = 128

Specifically, each DIMENSION statement must identify at least one unique HIERARCHY statement. The maximum number of hierarchies that can be defined in a dimension is determined only by the maximum number of hierarchies that can be

defined in a cube. All hierarchies other than the first hierarchy in each dimension apply to the total. Levels in the same dimension can be shared between hierarchies.

Defining the Number of Levels, Measures, Members, and Properties

The following are maximum values for the specified cube element:

Levels
: The maximum number of levels for a cube is 256. There can be up to 19 levels per hierarchy.

Measures
: The maximum number of measures per cube is 1024.

Members
: The maximum number of members is 2^32 members per hierarchy.

Properties
: The maximum number of properties per level is 256.

Naming Guidelines and Rules for the SAS OLAP Server

General Naming Guidelines

When you are creating SAS OLAP cubes, there are naming guidelines and restrictions that you should follow. Below are general guidelines for naming SAS OLAP cubes and related objects:

Name Uniqueness
: Name uniqueness is case insensitive. Therefore, a name can contain mixed-case letters. SAS stores and writes the variable name in the same case that is used in the first reference to the variable. However, when SAS processes a variable name, SAS internally converts it to uppercase. You cannot, therefore, use the same variable name with a different combination of uppercase and lowercase letters to represent different variables. For example, cat, Cat, and CAT all represent the same variable. Names must be unique within a folder by type but not by subtype.

Macro Variables or Functions
: When building or editing a SAS OLAP cube, you should be aware that the fields within the Cube Designer wizard do not support macro variables or functions. The fields are purely character strings and are not resolved before being written to the cube's metadata.

Name Literals

If the name has embedded blanks or characters other than letters of the Latin alphabet, numbers, or underscores, then PROC OLAP formats the name as a name literal. This means that it is enclosed within quotation marks followed by the letter n. (Name literals enable you to use special characters or blanks that are not otherwise allowed in SAS names. Here are some examples:

```
CUBE='Financials@HQ'n

DIMENSION 'Product@Work Dimension'n hierarchies=
(Product@Work Hierarchy'n)

HIERARCHY 'Product@Work Hierarchy'n levels=
(prodtype product)
```

VALIDVARNAME =Option

If you need to use special characters in a name, you must use the VALIDVARNAME= option. Ideally, the SAS Workspace Server should have the VALIDVARNAME= system option set to ANY. If the SAS Workspace Server is not running with VALIDVARNAME=ANY, then you can set the option on the Cube Designer wizard. In this case, select the **Advanced** button on the Cube Designer-General page. On the **Submit SAS Code** tab, enter the VALIDVARNAME= option. Modify your names to meet the naming requirements that the SAS Workspace Server is running with.

Note: For more information about the VALIDVARNAME= option, see "VALIDVARNAME= System Option" in the *SAS Language Reference: Dictionary*.

SAS OLAP Cubes

In addition to the general requirements, SAS OLAP Server names for cubes follow these general rules:

- A cube name can be up to 32 characters in length.
- A cube name must be a SAS name.
- A cube name cannot contain square brackets. This is an MDX restriction.
- A cube name cannot have non-printable characters.
- A cube name cannot have leading white spaces because cubes are public objects that are contained in metadata folders.
- A cube name cannot contain forward slashes or backward slashes.
- If you need to use special characters in a name, you must use the VALIDVARNAME=ANY option. This applies to special characters except forward slashes, backward slashes, and square brackets.

Dimensions, Levels, Hierarchies, and Measures

SAS OLAP Server names for dimensions, levels, hierarchies, and measures follow these general rules:

- A name can be up to 32 characters in length.
- A name must be a SAS name.
- If you need to use special characters in a name, you must use the VALIDVARNAME=ANY option.

Note: Cube dimensions cannot have the same name as any level name within the dimension. However, if there is only one hierarchy for the dimension it is automatically assigned the same name as the dimension.

OLAP Cube Aggregations

Aggregation names can be up to 32 characters in length. They can also include slashes, periods, and square brackets.

OLAP Schemas

SAS OLAP Server names for OLAP schemas follow these general rules:

- A schema name can be up to 60 characters in length.
- A schema name cannot have forward slashes or backward slashes.
- A schema name cannot have non-printable characters.
- A schema name cannot have leading white spaces because schemas are public objects that are contained in metadata folders.

Calculated Measures and Members

SAS OLAP Server names for calculated measures and members follow this general rule:

- A calculated measure or member name can be up to 60 characters in length. However, it is recommended that you maintain a 32-character limit for calculated measures.

SAS Formats Available for Measures

For each measure that you define for a cube, you can select the SAS format that is used to display the value of the measure. The following formats are available from the **Format** drop-down list on the Measure Details page of the Cube Designer wizard in SAS OLAP Cube Studio:

Table 5.1 *SAS Formats Available for Measures*

SAS Format	Description
15.0	Designates the width of the resulting string will be a maximum of 15 characters after the value has been formatted.
BEST	Enables SAS to choose the best numeric notation.
BEST15	Enables SAS to choose the best numeric notation. The width of the resulting string will be a maximum of 15 characters.
COMMA	Writes numeric values with commas and decimal points.
COMMA15	Writes numeric values with commas and decimal points. The width of the resulting string will be a maximum of 15 characters.
COMMAX	Writes numeric values with periods and commas.
COMMAX15	Writes numeric values with periods and commas. The width of the resulting string will be a maximum of 15 characters.
DOLLAR	Writes numeric values with dollar signs, commas, and decimal points.
DOLLAR15	Writes numeric values with dollar signs, commas, and decimal points. The width of the resulting string will be a maximum of 15 characters.
DOLLARX	Writes numeric values with dollar signs, periods, and commas.
DOLLARX15	Writes numeric values with dollar signs, periods, and commas. The width of the resulting string will be a maximum of 15 characters.

SAS Format	Description
EURO	Writes numeric values with a leading euro symbol, a comma that separates every three digits, and a period that separates the decimal fraction.
EURO15	Writes numeric values with a leading euro symbol, a comma that separates every three digits, and a period that separates the decimal fraction. The width of the resulting string will be a maximum of 15 characters.
EUROX	Writes numeric values with a leading euro symbol, a period that separates every three digits, and a comma that separates the decimal fraction.
EUROX15	Writes numeric values with a leading euro symbol, a period that separates every three digits, and a comma that separates the decimal fraction. The width of the resulting string will be a maximum of 15 characters.
NEGPAREN	Writes negative numeric values in parentheses.
NEGPAREN15	Writes negative numeric values in parentheses. The width of the resulting string will be a maximum of 15 characters.
PERCENT	Writes numeric values as percentages.
PERCENT15	Writes numeric values as percentages. The width of the resulting string will be a maximum of 15 characters. W.D writes standard numeric data one digit per byte.
WORDF	Writes numeric values as words with fractions that are shown numerically.
WORDF10	Writes numeric values as words with fractions that are shown numerically. The width of the resulting string will be a maximum of 10 characters.
WORDS	Writes numeric values as words.
WORDS10	Writes numeric values as words. The width of the resulting string will be a maximum of 10 characters.
YEN	Writes numeric values with yen signs, commas, and decimal points.
YEN15	Writes numeric values with yen signs, commas, and decimal points. The width of the resulting string will be a maximum of 15 characters.

Note: For more information about SAS formats, see "Formats" in the *SAS Language Reference: Dictionary*.

Statistics Available for Measures

When you define a cube, you select the statistics that are used to calculate the cube's measures. Here is the list of available statistics.

Base Stored Statistics:

- Count
- Sum
- Maximum
- Minimum
- Count of Missing Values
- Uncorrected Sum of Squares

Derived Statistics:

- Average
- Range
- Corrected Sum of Squares
- Variance
- Standard Deviation
- Standard Error of Mean
- Coefficient of Variance
- T Value
- Probability of Greater Absolute Value
- Lower Confidence Limit
- Upper Confidence Limit

New cubes that are based on a data source that contains existing summarized data must include measure statements for the stored statistics. Measure statements are required for each derived statistic that you want to create for the new cube. For example, if you want to calculate AVG, you must create measures for N and SUM, as well as AVG. The following table indicates which stored statistics are required for each derived statistic.

Table 5.2 *Stored Statistics Required for Derived Statistics*

Derived Statistics	Required Stored Statistics
AVG	N, SUM

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

6

Building Cubes and Administering Cubes

SAS OLAP Cube Studio and the OLAP Procedure

SAS OLAP Cube Studio

SAS OLAP Cube Studio contains several tools and functions that enable you build, modify, and maintain OLAP cubes. Specifically, the Cube Designer wizard guides you through the cube building process. You can specify the data used to build the cube, the structure of the cube, and the measures and aggregations that will be used to process queries for the cube. After you have built your cube in theCube Designer wizard, you can view a simple display of the cube and verify your cube data and structure.

SAS OLAP Cube Studio also provides tools and functions for modifying and maintaining cubes. After a cube is built you can make various changes and updates to the cube. Here are some functions that you can perform on a cube after it is created:

- tune the aggregations for the cube
- add and edit calculated members and measures for the cube

- incrementally update the cube
- export and then import a SAS package between SAS environments

For more information about functions and tasks that you can perform on a cube after it is created, see “Editing a Cube” on page 90 .

SAS OLAP Procedure

In addition to using SAS OLAP Cube Studio, you can define and build SAS OLAP cubes with the OLAP procedure. The OLAP procedure contains various statements and options to help you define a cube in batch mode if needed or preferred. See Chapter 12, “OLAP Procedure,” on page 315 for detailed information about the OLAP procedure.

This chapter discusses and refers primarily to building cubes in SAS OLAP Cube Studio. The comparable OLAP procedure statements and options are referred to where necessary in each topic.

Connecting and Reconnecting to a Metadata Server

Overview

The SAS Metadata Server enables users to write metadata objects to and read metadata objects from SAS folders. It maintains information about users, groups, data libraries, servers, and various user-created products such as cubes and information maps. The SAS Metadata Server contains a metadata identity for every user of the SAS Intelligence Platform. This includes each user's login information, including a user ID and an encrypted password. When a user logs on to a SAS application, the application verifies the user's identity by checking it against the metadata identity.

To create SAS OLAP cubes, you must create a connection profile for the metadata server that you access while using SAS OLAP Cube Studio. A connection profile is a user's view of a metadata server and is used to connect the application to a metadata server. The Connection Profile wizard enables you to specify the connection profile. When you define a connection profile, you specify the server and the port to which the application connects. You can also specify a user ID and password for all connections to the server or allow users to specify a user ID and password each time they log on. When an application opens a connection profile, it has access to the metadata on the specified server based on the security credentials of the user ID used to connect.

A connection profile is a client-side definition of where a SAS Metadata Server is located. The connection profile definition includes a host name and a port number,

and can contain a user's login information and instructions for connecting to the metadata server automatically.

Creating a Connection Profile

In order to define cubes, you must be connected to a running metadata server. Connection information is stored in a connection profile, which is a client-side definition of where the SAS Metadata Server is located. The profile definition includes a host name and port number. In addition, the connection profile can contain a user's login information and instructions for connecting to the metadata server automatically.

You can define new connection profiles using the Connection Profile wizard. This wizard is available from within various SAS products, including SAS OLAP Cube Studio, SAS Management Console, and SAS Data Integration Studio. To create a new connection profile, follow these steps:

1. In SAS OLAP Cube Studio, select **File** ⇨ **Connection Profile** to display the Connection Profile dialog box. The Connection Profile dialog box also appears automatically when you launch SAS OLAP Cube Studio, unless you have set a default profile.
2. Select the **Create a new connection profile** radio button to launch the Connection Profile wizard.
3. On the Connection Profile wizard, enter connection information to create the new profile, including the following items:
   - profile name
   - machine name
   - port number
   - user name
   - password
   - authentication domain
   - whether to save user ID and password in this profile
   - whether to use Integrated Windows authentication (single sign-on)
4. Click **Finish** to return to the Connection Profile dialog box.
5. If you want to always use the selected profile when SAS OLAP Cube Studio is launched, click the **Set this connection profile as the default** check box.
6. Click **OK** to open the selected profile.

Connecting with an Existing Profile

Existing connection profiles are listed on the Connection Profile dialog box, which appears automatically when you launch SAS OLAP Cube Studio (unless you have

set a default profile). If the dialog box does not appear automatically, select **File** ⇨ **Connection Profile**. Complete these steps in the Connection Profile dialog box:

1. Select the **Open an existing connection profile** radio button.
2. Select the name of the profile from the drop-down selection list.
3. If you want to always use the selected profile when SAS OLAP Cube Studio is launched, select the **Set this connection profile as the default** check-box.
4. Click **OK** to open the selected profile.

Reconnecting to a Metadata Server

If the connection to the metadata server is broken, a dialog box appears and asks whether you want to attempt reconnection. Click **Try Now**. SAS OLAP Cube Studio then attempts to reconnect to the metadata server. If the reconnection is successful, you can continue your work. The user credentials from the previous session are used.

Cube Designer Wizard

Within SAS OLAP Cube Studio, the Cube Designer wizard guides you through the process of building a cube. You can create cube definitions as well as build complete cubes. The following are some tasks that you can perform:

- define the data sources used to load a cube
- specify any drill-through tables used by the cube
- define the cube dimensions, levels, and hierarchies
- select measures and measure details for the cube
- specify unique member count measures (NUNIQUE)
- specify member properties
- configure aggregations
- select language translation tables for a star schema's dimension tables
- define user-written formats to create the cube
- define how to handle missing values for ragged hierarchies

Defining Data Sources for a Cube

When you create a cube, you must specify the data source tables for the cube. Those data source tables must be registered in the SAS metadata. Within SAS OLAP Cube Studio, the Register Table wizard enables you to identify a data source and register it in the SAS metadata. Any data source that you plan to use for your cube must be registered in the SAS metadata so that it is accessible in SAS OLAP Cube Studio.

The type of data table or tables that you are using determines the type of cube that you can build. There are three types of SAS OLAP cubes that you can build:

1. detail table
2. star schema
3. fully summarized table

On the Cube Designer – General page, you first specify the cube name, the storage location for the cube, and the type of input data that you are using. The type of cube that you create (detail, star schema, fully summarized) determines the information that you must specify in the Cube Designer wizard. A star schema cube requires different information than a detail cube or a fully summarized cube. As a result, the wizard presents a slightly different arrangement of steps for each type of cube that you build.

After you complete entering information for the Cube Designer – General page, you then select the specific input data table for the cube. All cube types require an input data table. The Cube Designer – Input page enables you to select a datasource table for the cube. You can also view the underlying table data and the table properties from this page. Star schema and fully summarized cubes have additional tables that must be specified as follows:

- For star schema cubes, you must also define the dimension tables for the cube. Dimension tables are defined after input tables on the Cube Designer – Dimension Tables page.
- For fully summarized cubes, you also define aggregation tables on the Cube Designer – Aggregation Tables page. You do this when you define aggregations for the cube.

Defining Drill-Through Tables

Drill-through tables enable you to display, at query time, the unsummarized detail data that a table cell or selected cells were summarized from. Using the drill-through capability, you can view the specific observations from the underlying data that make up an aggregate value. Many OLAP applications give you the ability to select a cell or a range of cells and then view the input data that the cell data were

summarized from. Drill-through capability enables companies to access data that is not stored on an OLAP server and make it accessible to end users of an OLAP application. When the OLAP server receives a request for this additional data at query time, it automatically submits a query and retrieves the data from a data warehouse or from an OLTP (online transaction processing) system. In order to provide this capability, a drill-through table must be specified for a cube.

A drill-through table can be a view, data set, or other data file that contains data that is used to define a cube. For you to successfully use a drill-through table, it must have the same columns with the same attributes as the table that the cube was loaded from. Detail data tables are most commonly used as drill-through tables. However, if the cube was loaded from a star schema, a view that fully joins the star schema can be used as a drill-through table.

In the Cube Designer wizard, you can select a drill-through table on the Cube Designer – Input page. Select a table from the Available tables list and move it to the Selected table list.

You can also use the OLAP procedure, DRILLTHROUGH_TABLE | DT_TABLE | DT_TBL= option to define a drill-through table for a cube. To understand how drill-through is implemented in a SAS OLAP cube, see the cube building example “Implementing Drill-through to Detail Data in a SAS OLAP Cube” on page 272.

Note: When you select a data table for drill-through, you might need to consider user access and security restrictions for that table. For further information, see “Security for Drill-through Tables” on page 126.

Defining Dimensions and Levels

Overview

Within the Cube Designer wizard, you can specify dimensions and levels for a cube on the Dimension Designer wizard. This wizard enables you to select the type of dimension that you are creating and then select the levels for that dimension.

Defining Dimensions

The Dimension Designer wizard enables you to add new dimensions to a cube and edit existing dimensions. On the Dimension Designer – General page, you can specify the following information for a dimension:

- the type of dimension that you are creating (STANDARD, TIME, or GEO)
- the sort order for level values in that dimension
- whether to allow new members for the dimension during an incremental update of the cube

- star schema mapping details. such as the dimension table, key, and fact key
- missing values for ragged or unbalanced data

Defining Levels for a Dimension

After you specify the general information for a dimension, you can select the levels for that dimension. On the Dimension Designer – Level page, select the **Add** button. The Add Levels dialog box appears. From here you can select the input columns for the levels that you are adding to the dimension. You can have a maximum of 256 levels per cube.

If you have defined a dimension as a TIME-type dimension, the **Add** button is changed to a drop-down list of options. A second option for adding levels to the dimension is then available. You can select either of the following options:

- **Add levels**
- **Add supplied time hierarchies**

Note: For more information, see "Specifying the TIME Dimension" on page 66.

After you have selected the levels for a dimension, you can select the attributes for each level on the Dimension Designer – Level page. You can specify the following information for a level:

- format
- caption
- sort order
- missing member string
- description

You can also add more levels to the dimension, duplicate a level, or delete a level.

Specifying the TIME Dimension

Designating a TIME Dimension

The TIME dimension is a unique type of dimension for SAS OLAP cubes. When building a cube, and specifically, when defining a dimension, you must select the type of dimension that you are creating. You can create STANDARD-, TIME-, or GEO-type dimensions. For the TIME dimension, you can also provide time hierarchies with the input table for the cube. Supplied time hierarchies can help you build the dimension and auto-populate the levels and some level properties for the cube.

On the Dimension Designer – General page select the TIME type dimension. On the Dimension Designer – Level page, the **Add** button becomes a drop-down list of options. The **Add levels** and **Add supplied time hierarchies** options are now available for selection.

Note: The Add (levels) function is always available for selection regardless of the type of dimension that you are creating. When you specify a TIME-type dimension, the **Add** button is converted to a drop-down list. where you can select which of the Add (level) specific functions that you can use.

Add Levels

The Add Levels dialog box enables you to select input columns for the levels that you are adding to the dimension. When you select the **Add levels** option, the Add Levels dialog box appears. You can select from a list of available input columns and add them to the list of input columns for new levels.

Add Supplied Time Hierarchies

In the Dimension Designer, the **Add supplied time hierarchies** option becomes available when a dimension is defined as a TIME-type dimension and supplied time hierarchies are available. When you select the **Add supplied time hierarchies** option, the Add Supplied dialog box appears. For time-specific dimensions, you can select from a list of supplied time hierarchies. These supplied time hierarchies can help you build the dimension and auto-populate the levels and some level properties for the cube. In this dialog box, you select from a list of supplied time hierarchies for the dimension. The Add Supplied dialog box is available only if the TIME type is selected on the Dimension Designer – General page.

Note: For more information about applying time hierarchies, see the cube building example “Creating a TIME Dimension in SAS OLAP Cube Studio” on page 258.

Specifying GIS Map Information for a Dimension

Overview

The SAS OLAP Cube Studio GIS Map function enables you to store Esri Geographic Information System (GIS) spatial map information in the SAS metadata. This GIS information can then be read by the SAS OLAP Server and returned during a cube query.

When building or editing a cube, the GIS Map function enables you to identify a geography-based dimension and then assign Esri spatial map information to that dimension. To define GIS information for a SAS OLAP cube, you identify a dimension as a geographic-type dimension (GEO) in the Dimension Designer – General window. You can have only one GEO dimension per cube. After a dimension has been marked as a GEO-type dimension, the **GIS Map** button becomes active on the Cube Designer – Dimensions page.

The GIS Map dialog box enables you to assign Esri spatial map information to levels of the GEO-type dimension. You can add map information to an existing cube, modify map information for a cube, or delete the map information from a cube. When Esri map information is added to a cube, the property objects for the mapped cube levels are listed in the Cube Designer's Member Property window. From here you can modify the format, caption, and description for the member property. These member properties are named SAS_SPATIAL_ID by default.

You can set up the Esri ArcGIS server information in the metadata by using the Map Service Manager plug-in in SAS Management Console. Users can access the functionality through the SAS Web OLAP Viewer.

Note: If a dimension is changed from type GEO to STANDARD or TIME, all the map information is removed.

For further information about the GIS Map window, see the SAS OLAP Cube Studio Help topic “GIS Maps.” For information about the Esri Map Component, see the “Configuring the Esri Map Component” chapter in the *SAS Intelligence Platform: Web Application Administration Guide*.

Specifying a GEO-Type Dimension in PROC OLAP

If you create a cube with the OLAP procedure, you can define a cube dimension as a GEO type. The TYPE= option, in the DIMENSION statement, enables you to identify a dimension as a GEO-type dimension. This can be used only when you initially create a cube with the OLAP procedure. You can add geographic

information to an existing cube using SAS OLAP Cube Studio or with the PROC OLAP options ESRI_MAP_SERVER, ESRI_MAP_LAYER, and ESRI_MAP_FIELD. For more details, see the TYPE= option.

Defining Cube Hierarchies

Defining a Default Hierarchy

When you define cube dimensions, levels, and hierarchies in SAS OLAP Cube Studio, a default hierarchy for a dimension is automatically created if a hierarchy is not explicitly defined. This default hierarchy includes all levels that were specified for the current dimension and the order that they were listed in for the dimension. In addition, if you define multiple hierarchies and do not select a default, then the default is automatically assigned to the first hierarchy that is created for the dimension. On the Dimension Designer – Hierarchy window, you can click the **Default** button to set a selected hierarchy as the default for the dimension.

Defining Multiple Hierarchies for a Dimension

Overview

SAS OLAP cubes are organized into dimensions and levels of data. The levels are then arranged into hierarchies. After an initial hierarchy has been created, you can define additional hierarchies for a single dimension of a cube. This enables you to have multiple possible drill paths for the same data. When you create more than one hierarchy for a dimension, the levels have some restrictions:

- A level in a dimension might be used in more than one hierarchy within that dimension. However, levels cannot be used in hierarchies that are not defined within the dimension that the level is defined in.
- Each level must be used in at least one hierarchy.
- Levels from the same dimension that are picked for an aggregation must be in the drill order for at least one hierarchy in that dimension.
- You cannot share levels between dimensions.

You can arrange the levels in a hierarchy in any order. The one exception to this is the TIME dimension. Levels in hierarchies in the TIME dimension must follow a prescribed order that is determined by the numerical value that is assigned to the type. This order is from the smallest value (Years, 16) to the greatest value (Seconds, 3,096). You can have only one TIME dimension for a cube. The dimension hierarchies also have some restrictions:

- The first hierarchy that is defined for the dimension is designated as the default. When there are multiple hierarchies, you can designate the default hierarchy for the dimension.
- Hierarchy names must be unique across the cube. If there is a single hierarchy for a dimension, then its name must be the name of the dimension. Also, dimension and hierarchy names cannot be the same as a level name within that dimension.
- For any cube that is loaded with a star schema, in which a dimension table represents multiple hierarchies for that dimension, the dimension key that is used to join the dimension table to the fact table will be used for all hierarchies of that dimension.

HIERARCHY Statement

The HIERARCHY statement is used with the PROC OLAP statement when you define a cube:

```
hierarchy campaigns
   levels=(campaign_type campaign sub_campaign);
```

Cube Designer

You can establish multiple hierarchies by using the Cube Designer – Dimension page, which is located in the SAS OLAP Cube Studio Cube Designer. To add a hierarchy to an existing dimension, select a dimension, and then click **Modify**. This opens the Dimension Designer – General page. It is populated with the values for the selected dimension. Select **Next** until you reach the Dimension Designer - Hierarchy page. Select **Add** to create an additional hierarchy.

Note: You can modify existing hierarchies by selecting a hierarchy and clicking **Modify**. You can also assign a default hierarchy by selecting a hierarchy and clicking **Default**. The first hierarchy is automatically the default hierarchy.

Note: An exception to defining multiple hierarchies for a dimension is the TIME dimension. Levels in hierarchies in the TIME dimension must follow a prescribed order that is determined by the numeric value that is assigned to the type. This order is from the smallest value (Year, 16) to the greatest value (Seconds, 3,096).

On the Dimension Designer – Define a Hierarchy page, you can define a new hierarchy and select the different levels and their order for the hierarchy.

Defining Ragged and Unbalanced Hierarchies for a Dimension

Overview

Dimension levels are arranged in one or more hierarchies. Hierarchies, by process of ordering, have a branching arrangement, and the different member levels have parent and child relationships. For example, at company X the sales staff are located in different regions and cities in different countries. A balanced hierarchy might look like this:

- global sales president (top of hierarchy)
- sales presidents (per country)
- regional sales managers
- city sales managers

Because of differences in the cube data, hierarchies are often not balanced and possibly have missing members. For example, some sales regions might not have sales managers assigned to a specific city. Or some countries might not have sales regions, just cities. These real-world scenarios would create hierarchies that have missing member data and possibly ragged hierarchies. This affects the drill path of the cube data.

You can also drill to missing members within a path and continue to drill down to members that are present.

Note: Existing SAS 9.2 OLAP cubes that have been updated with one or more new members for a hierarchy can possibly contain ragged hierarchies and must be rebuilt in SAS 9.4.

Defining Ragged and Unbalanced Hierarchies in SAS OLAP Cube Studio

The Cube Designer in SAS OLAP Cube Studio enables you to specify the missing members for a hierarchy and the type of data that is missing. Here are the Cube Designer pages that enable you to specify missing member information:

Ragged Hierarchies
: Located on the Advanced Cube Options dialog box, on the Cube Designer – General page, this tab enables you to specify character and numeric missing member information. By default, no missing member information is indicated with the value **None**.

Dimension Designer – General – Advanced dialog
You can select one of these options in the Advanced dialog box:

- Inherit cube settings for missing member strings.
- Skip over missing members with this character string (override cube settings). For this dimension only, enter a maximum of 256 characters that will be used to identify missing character members.
- Skip over missing members with this numeric string (override cube settings). For this dimension only, enter a maximum of 256 characters that will be used to identify missing numeric members.

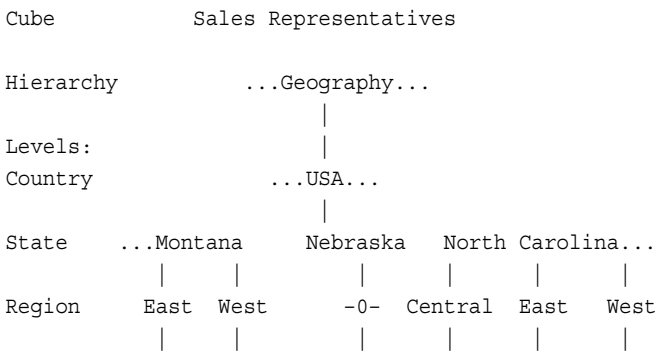
Dimension Designer – Level Properties
You can select one of these options for each level:

- Inherit the dimension setting for handling missing members.
- Ignore the inherited string that is defined in the dimension for handling missing members.
- Use a blank space to identify missing members.
- Use a period to identify missing numeric members.
- For this level, enter a maximum of 256 characters that will be used to identify missing members.

Defining Ragged and Unbalanced Hierarchies with PROC OLAP

To create ragged and unbalanced hierarchies with PROC OLAP, you specify options that allow the procedure to skip over members of levels that have captions with specified values. The presence of these skipped members constitutes a ragged or unbalanced hierarchy. In a ragged hierarchy, skipped members in a given level can have descendants; skipped members are used to enable drill-down through empty levels. In an unbalanced hierarchy, the skipped members do not have descendants; members are skipped in order to create hierarchies where certain branches do extend to all available levels.

For an example of the creation of a ragged hierarchy, assume that a cube defines information about sales representatives. The Geography hierarchy is defined to have the levels Country, State, Region, and City. In this particular sales organization, the state of Nebraska has no regions, but it does have sales representatives in a number of cities. This ragged hierarchy can be shown as follows:



```
City        |  Billings...  |       |       |      Asheville...
            |               |       |    Greenville...
          Helena...         |   Charlotte...
                            |
         ...Council Bluffs, Lincoln, Sioux City...
```

The Geography hierarchy is ragged because of the need to skip the Region level, and because the skipped member (Nebraska) has descendants.

To create the ragged hierarchy shown above, the Nebraska member needs to be defined with one member at the Region level. That member needs to have a caption that matches the value of the EMPTY_CHAR= option that is defined in the respective DIMENSION statement.

In the resulting cube, drilling down from Nebraska takes you directly from the State level to the City level.

For an example of the creation of an unbalanced hierarchy, assume that a cube named Employees has a hierarchy named Organizations. In that hierarchy there are various divisions, departments, and groups. As shown below, some departments lack groups:

```
Cube                 Employees

Hierarchy         ...Organizations...
                            |
Levels:                     |
Div     ...R&D          Sales     Tech Support...
             |              |            |
Dept     Java  Platform     |   Java Support  Platform Support
             |     |        |            |       |
Group        |     |        |           -0-     -0-
             |     |        |
             |     |    Global    Marketing    U.S.
             |     |
             |    Hosts    Performance    Unit Test
             |
          Client    Mid-Tier    Server
```

The preceding hierarchy is unbalanced because the Tech Support level has no descendants at the Group level. To implement this unbalanced hierarchy, the levels Java Support and Platform Support would have to be defined with captions that matched the value of the EMPTY= option in their respective LEVEL statements.

The options that implement ragged and unbalanced hierarchies are found in the PROC OLAP statement, DIMENSION statement, and LEVEL statement. In the PROC OLAP and DIMENSION statements, you can specify separate caption values for character and numeric levels using the options EMPTY_CHAR= and EMPTY_NUM=. Similarly, the EMPTY= option of the LEVEL statement enables you to specify separate values for each level in a hierarchy, regardless of any similar values that were specified in preceding DIMENSION and PROC OLAP statements.

The DIMENSION and LEVEL statements also provide the IGNORE_EMPTY option, which specifies that any prior specifications of EMPTY_CHAR= or EMPTY_NUM= are to be ignored for that hierarchy or level.

Ragged Hierarchies and Unique Member Names

In a ragged hierarchy, the parent of a member might not be at the level directly above that member. Furthermore, not all children of a member are necessarily at the same level. This can lead to a situation where two children have the same unique name.

For example, in a geography hierarchy, you might have the levels state, county, and city. The state Washington might have a child at the county level called Olympia and another child at the city level, also named Olympia. The city member is not a descendant of the county member of the same name. It is a child of Washington.

In a ragged hierarchy, levels can have an unconventional structure, and unpopulated levels are not assigned a token or placeholder. As a result, the unique name for the county member is `Geography.[All Geography].Washington.Olympia`, and the unique name for the city member is `Geography.[All Geography].Washington.Olympia`.

The result of this anomaly is that the city member cannot be asked for by a unique name in a query, either through MDX or an OLE DB for OLAP (ODBO) request for metadata. It will be returned in any set that contains it so the data that is associated with it is not lost. The same applies to the children of a member such as Olympia. Because the server searches through the hierarchy to validate member names, a request by name for a child of Olympia the city will result in a bad member name error. This is because the server actually searches under the county Olympia.

This situation occurs only when two members with the same name share a parent. Any number of members named Olympia could exist under other parents with no unusual results.

MDX Function Behavior and Ragged or Unbalanced Hierarchies

Below is a table that describes the behavior of various MDX functions when used with ragged or unbalanced hierarchies.

MDX Function	Behavior When Used with a Ragged or Unbalanced Hierarchy
AllMembers Members	Ignores empty positions in the hierarchy, except for calculated members that are added below the leaf level of an unbalanced hierarchy. In SAS 9.1, those members are irretrievable.
Children	Returns real members from the level below the member, as well as the members below

MDX Function	Behavior When Used with a Ragged or Unbalanced Hierarchy
	any virtual members. This behavior is recursive.
Descendants	Returns empty if the level name is passed. If distance n is passed, it returns the member from n populated steps away.
DrillDownLevel DrillDownLevelTop DrillDownLevelBottom DrillUpLevel	Continues seeking members at lower levels when virtual members are encountered.
DrillDownMember DrillDownMemberTop DrillDownMemberBottom DrillUpMember	Skips empty positions in the hierarchy and returns the first real member.
LastPeriods YTD QTD MTD WTD	Returns real members in the result set. Empty if all are virtual.
AddCalculatedMembers	In SAS 9.1, calculated members added below leaf level in unbalanced hierarchies are irretrievable.
Parent Ancestor FirstChild LastChild	Skips empty positions in a hierarchy.
ClosingPeriod OpeningPeriod Cousin ParallelPeriod	Returns an empty member if a virtual member is in a requested position.
PrevMember NextMember Lead Lag	Ignores virtual members.
Siblings	Returns real member(s) at level.

MDX Function	Behavior When Used with a Ragged or Unbalanced Hierarchy
FirstSibling	
LastSibling	

Defining Measures for a Cube

Selecting Measures in SAS OLAP Cube Studio

When you build a cube in the Cube Designer wizard, you must specify the measures that are used to help query the cube. After you have defined the data input source and the structure of the cube (dimensions, levels, hierarchies) you can define measures on the Select Measures page of the Cube Designer wizard. You can view the cube's currently defined measures, add new measures, and remove existing measures.

To define a measure, you select from the **Available** list of columns and statistics. This list contains the numeric columns in the input data source and the statistics available for each column. Move the selected items to the **Cube Measures** table. You can have a maximum of 1024 measures per cube.

Note: If you are including aggregated data from tables other than the input data source, considerations for stored statistics should be made. This applies when you are creating a cube from a detail table or a star schema. You must include measures for the stored statistics that are required for each derived statistic that you create in the cube. For example, if you want to calculate AVG, you must create measures for N and SUM, as well as AVG.

Unique Member Count Measures

You define unique member count measures in the Cube Designer – Select Measures page. Select the level and hierarchy for the new measure, and then move them to the **Cube Measures** table. A name is automatically generated for that unique member count measure consisting of the level name,"NUNIQUE", and the parent hierarchy name. It is important to note that only one combination of level and hierarchy can be defined for a measure. After a combination has been used to create a unique member count measure, that combination cannot be used again. In addition, if the level or hierarchy is deselected, then any associated defined unique member count measures are deleted from the cube.

For a cube to build, at least one non-unique member count (non-NUNIQUE) measure must be defined for the cube.

NUNIQUE Statistic

You can define a distinct count statistic using the MEASURE statement and the NUNIQUE statistic. The LEVEL and HIERARCHY options for the MEASURE statement are used with the NUNIQUE statistic and are ignored for non-NUNIQUE statistics if specified.

Note: The LEVEL name is optional. If it is omitted, then the level name is assumed to be the name specified for the NUNIQUE measure. The HIERARCHY name is required only if the level is in multiple hierarchies. For further information about using the NUNIQUE statistic see the MEASURE Statement.

Defining Stored and Derived Measures for a Fully Summarized Cube

Overview

When you are building a cube with fully summarized data, you can select predefined (stored) measures that are actually stored with the summarized data source. You can also then create derived measures from those stored measures. In the Cube Designer wizard, pages for stored and derived measures are available when you have selected a fully summarized data source to build a cube from.

Stored Measures

On the Stored Measures page of the Cube Designer wizard, you can select the NWAY columns that contain stored measures data. Select from the **Available** list and move the needed measures to the **Selected** list. On the Assign Stored Measures page of the Cube Designer wizard, you can then assign a statistic to each stored measure and each measure to an analysis group.

Derived Measures

After you have selected stored measures from the summarized data source, you can define any needed derived measures for the cube. You can store only derived statistics that can be calculated from the available stored statistics. For example, to

use the derived statistic AVG, you must have stored statistics for N and SUM with the same assigned group name.

On the Select Derived Measures page of the Cube Designer wizard, select the **Add** button to create a new derived statistic or the **Modify** button to change an existing derived measure. To define a derived measure, you specify the following options:

Analysis Group
displays a list of analysis groups assigned to the cube's stored measures. Select a group from this drop-down list to see available statistics in the **Derived Statistics** drop-down list. Only those groups that contain stored measures that can be used to derive a statistic are included in the drop-down list.

Derived Statistics
displays a list of statistics that you can derive from the statistics that you are storing with the cube for the selected analysis group. Select the derived statistic that you want to use. Only those statistics that can be derived from the stored statistics are included in the drop-down list

Measure Name
displays a name for the derived measure. The name must be a valid SAS name (up to 32 characters). For more information, see “Naming Guidelines and Rules for the SAS OLAP Server ” on page 51.

Note: A derived statistic can be used only once per cube per analysis group.

Statistics Available for Measures

Overview

When you define a cube, you select the statistics that are used to calculate the cube's measures. Here is the list of available statistics.

Base Stored Statistics

- Count
- Sum
- Maximum
- Minimum
- Count of Missing Values
- Uncorrected Sum of Squares

Derived Statistics

- Average
- Range
- Correct Sum of Squares
- Variance
- Standard Deviation
- Standard Error of Mean
- Coefficient of Variance
- T Value
- Probability of Greater Absolute Value
- Lower Confidence Limit
- Upper Confidence Limit

Required Measure Statements for Derived Statistics

New cubes that are based on a data source that contains existing summarized data must include measure statements for the stored statistics, which are required for each derived statistic that you want to create for the new cube. For example, if you want to calculate AVG, you must create measures for N and SUM, as well as AVG. The following table indicates which stored statistics are required for each derived statistic.

Table 6.1 *Stored Statistics That Are Used to Create Derived Statistics*

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

Defining Member Properties

Introduction

When you create a SAS OLAP cube, the information that is relevant to the cube is defined with the cube hierarchy, measures, and aggregations (summaries) that will be stored with the cube. Additional information that is part of the cube member data can be included in the cube definition as a member property. Member properties are attributes of dimension members that provide an additional gradation of information to users of the cube data. Member property information is usually not as significant as the levels and members within a dimension, and therefore does not qualify as a level or member. However, it can have additional analytical value that can be useful at query time.

A member property is assigned to a level within a hierarchy, and a level can have multiple properties that are assigned to it. For hierarchy placement, a member property is assigned (by default) to all hierarchies that the select level is included in. However, you can remove one or more (but not all) of the hierarchies that the member property is assigned to. When you create a member property, you must specify the property name, column, and level. Member property names can be shared across a cube but must be unique for a specific level within a specific hierarchy. You can also specify a caption, description, and format. The format that you specify here is used instead of the format in the data set.

Property Statement

The PROPERTY statement is used with the PROC OLAP statement when you define a cube:

```
PROPERTY zipcode-region
   column=post_code
   hierarchy=geographic
   level=region;
```

Cube Designer

You can also establish member properties with the Member Property dialog box that is part of the Cube Designer wizard in SAS OLAP Cube Studio. The Member Property dialog box is accessed from the Cube Designer – Dimensions page when you create a cube or edit a cube. On the Dimensions page, select **Add Member Property** to create a member property. In the Define a Member Property dialog box, enter the member property name, level, column, and caption.

Defining Aggregations While Building a Cube

Adding Aggregations to a Cube

The Cube Designer wizard enables you to add aggregations to a cube when the cube is created. You can define aggregations that are built with the cube on the Aggregations page of the Cube Designer wizard. From here you can add and modify aggregations, specify performance options, and choose whether to create the default NWAY aggregation. Selecting the **Add** button opens the Add Aggregation page of the Cube Designer wizard. It enables you to select from available levels to create an aggregation.

Defining Stored Aggregations for a Fully Summarized Cube

You can also specify stored aggregations for a cube. When you are building a cube in the Cube Designer wizard, you can choose to build from a detail table, a star schema, or a fully summarized table. If you build a cube from a fully summarized table, you can automatically specify aggregations that are stored in aggregation tables. If you are building a cube from a detail table or a star schema, you can also select to include aggregations that are stored in aggregation tables. The General page of the Cube Designer wizard provides the check-box option **Cube will use aggregated data from other tables**. This option enables you to create your cube using aggregated data from tables other than the detail table or star schema that you selected as the input.

The Aggregation Tables page of the Cube Designer wizard enables you to select aggregation tables for the cube. From these aggregation tables, you can then select stored aggregations to include with the cube. On the Stored Aggregations page of the Cube Designer wizard, you can add and modify a stored aggregation. After you have defined the stored aggregations, you can affix further aggregations that are built with the cube.

Saving the Cube Metadata or Creating the Physical Cube

Introduction

When you are building a cube in the Cube Designer wizard, you can choose to save only the cube's definition to the active metadata or to save the cube's definition and build the cube. Your options depend on the following considerations:

- Is the cube new?
- Is the cube defined only in the metadata?
- Does the cube physically exist?

New Cubes

If you are creating a new cube, the Finish page of the Cube Designer wizard contains these options:

Build physical cube now
builds the cube on the specified OLAP server.

Save the cube definition
registers only the cube definition to the current metadata repository, without building the physical cube. Use this option if you want to build the cube at a time of lower demand on the cube or the OLAP server.

Cubes Only Defined in the Metadata

If you are editing a cube that exists only in the metadata, the Finish page contains these items:

Build physical cube now
builds the cube on the specified OLAP server.

Save the cube definition
registers only the cube definition to the current metadata repository, without building the physical cube. Use this option if you want to build the cube at a time of lower demand on the cube or the OLAP server.

Cubes That Physically Exist

If you are editing a cube that physically exists, the Finish page contains these items:

Build physical cube now
saves definition changes to the current metadata and creates the physical cube. Any existing physical cube is deleted before the new cube is built. If the cube deletion fails, then the cube metadata is not updated.

Save the definition and delete the cube's physical data
registers only the modified cube definition in the current metadata. Any existing physical cube is deleted, but the modified metadata is not used to build a new cube. If the deletion fails, then the cube metadata is not updated. You can use this option to delete a physical cube without changing the cube's definition; simply navigate to the Finish window without making any changes to the cube's definition.

Note: If a cube uses a shared dimension and the shared dimension has not been built, the only option on the Finish page is **Save the cube definition**.

Overview

The **Export Code** button opens the Export Code dialog box. Click **Finish** to process the cube and close the Cube Designer wizard. After processing, a standard SAS log window displays the short form of PROC OLAP that was submitted to create the cube.

Note: To perform tasks on the physical cube (such as deleting, rebuilding, and tuning), you must have the appropriate file access permissions at the operating system level. If you do not have access, contact your system administrator for more information.

Saving the OLAP Procedure (Long Form versus Short Form)

Overview

The OLAP procedure is one of the SAS tools that you can use to create, update, and delete cubes. It provides statements that enable you to add dimensions, levels, hierarchies, measures, and aggregations to a cube as it is being created. These statements, when combined with the PROC OLAP statement and METASVR statement, are often referred to as the long form of PROC OLAP. The long form of PROC OLAP is used when there is no metadata definition for a cube in the SAS metadata. When submitted, the long form of the procedure first creates the metadata definition of the cube in the SAS metadata and then creates the physical files for the cube. If a metadata definition for the cube specified in the PROC OLAP statement already exists, an error message is printed in the log.

In addition to basic cube creation tasks, you can use a shorter form of PROC OLAP to perform maintenance tasks on an existing cube metadata definition. The short form of PROC OLAP must include the PROC OLAP statement and the METASVR statement. Other statements, such as the DEFINE, AGGREGATION, DROP_AGGREGATION and UNDEFINE statements can also be included, depending on the task that you want to perform. These include the following:

- delete the physical files for the cube while leaving the metadata definition intact (PROC OLAP with the DELETE_PHYSICAL option and METASVR statements)
- create the physical files for the cube based on the information in the existing metadata definition (PROC OLAP and METASVR statements)
- define new calculated members for a cube or remove existing calculated members (PROC OLAP, METASVR and DEFINE or UNDEFINE statements)
- define new aggregations or delete existing aggregations (PROC OLAP, METASVR, AGGREGATION or DROP_AGGREGATION statements)

You can use SAS OLAP Cube Studio to modify the metadata definition of a cube and then use the short form of PROC OLAP to rebuild the cube, including the changes. For example, if you want to add a dimension to an existing cube, edit the cube structure in SAS OLAP Cube Studio to add the dimension. Then, from the Finish panel of the Cube Designer, submit the short form of PROC OLAP to delete the physical files for the cube, save the changes to the metadata for the cube, and then submit the short form of PROC OLAP to re-create the physical files. An advantage to using the short form in this fashion is that links to access control entries, information maps, and reports will not be broken as the cube is updated.

To accomplish the same task using the long form of PROC OLAP, you would still use the short form with the DELETE option to delete the physical cube files and to delete the metadata definition. The appropriate DIMENSION statements would be added to the long form and the code submitted to create the new metadata and the new physical files. But now the access controls for the cube would need to be re-

created and information maps and reports would need to be adjusted to account for the newly created cube.

Export Code

The Export Code function is available in SAS OLAP Cube Studio. The Export Code function enables you to store in a text file the PROC OLAP code that is used to edit or build the cube. There are two options for exporting the PROC OLAP code: the long form or the short form. The Export Code dialog box appears when you perform one of these steps:

- Click **Export Code** on the Finish page in the Cube Designer wizard.
- Select **Export Code** from the **Actions** menu.

On the Export Code dialog box, you can select one of the following options:

1. **Long form** – enables you to save the full version of the OLAP procedure code to a text file.
2. **Short form** – enables you to save the short version of the OLAP procedure code to a text file.

The Export Code function is also available from the Aggregation Tuning and Incremental Update functions. When accessed from these functions, the text file that is generated contains the short form of PROC OLAP.

For both the long form and short form options, enter a fully qualified path, including file extension, to the location for the text file. The text file that is created stores the PROC OLAP code that is used to create the currently selected cube. Use the **Browse** button to navigate to a location on the file system. You can also open the drop-down box for either the long or short form options. SAS OLAP Cube Studio maintains a record of the five most recently used files for you to select from. The most recently used file is at the top of the list.

The PROC OLAP code that is generated can contain MDX expressions. If the MDX expression contains mixed quotation marks (both single and double quotes), do not place quotation marks around the MDX string. You must resolve this issue before submitting the PROC OLAP code. When working with quotation marks follow these guidelines:

- If the MDX string contains only single quotation marks, then place double quotation marks around the MDX string.
- If the MDX string contains only double quotation marks, then place single quotation marks around the MDX string.

Note: The PROC OLAP code that is generated by the Export Code function does not include the user ID and password due to security reasons. You must manually add the user ID and password to the exported file before submitting.

Viewing Cubes in SAS OLAP Cube Studio

Overview

After you have built a cube in SAS OLAP Cube Studio, you can examine the contents of the cube with the View Cube function. The View Cube function is available from the **Actions** menu and from the cube context menu. The View Cube function is available only for use with physically built cubes.

SAS OLAP Server Connection

When you select the View Cube function, a connection to the SAS OLAP Server must be made. You can identify the SAS OLAP Server for a cube by selecting the cube's OLAP schema in the tree view. Then select the **Server Assignment** tab on the Properties dialog box for that schema. If there is only one SAS OLAP Server assigned to the OLAP schema, then that server is used. If a SAS OLAP Server is not selected for the schema, a dialog box is displayed that enables you to select a SAS OLAP server. If a connection to the SAS OLAP Server cannot be made, then the View Cube dialog box cannot be launched.

Cube Permissions

In order to use the View Cube function, a cube must have the appropriate Read and ReadMetadata permissions. Read and ReadMetadata permissions can be granted for the cube with the Authorization function that is available in the Properties dialog box. If you do not have permissions to read the cube, the View Cube function fails when you attempt to load it. The following authorization error message is displayed:

```
View Cube results are not available. Possible reasons are
1) an error occurred in reading the data or
2) the permissions may not allow viewing
```

You can select **OK** to close the message dialog box, or you can select **Details**. The Details panel displays the following message:

```
The set is empty
```

In addition, if the Read permissions for a component of the cube (dimension, hierarchy, and so on) have been denied, those components are not available for selection in the cube dimension tree.

Disabled Cubes

If a cube has been disabled, you can see the same restriction as if the cube has been denied Read permissions. The View Cube function fails when you attempt to load it. The following authorization error message is displayed:

```
View Cube results are not available. Possible reasons are
1) an error occurred in reading the data or
2) the permissions may not allow viewing
```

You can select **OK** to close the message dialog box, or you can select **Details**. The Details panel then displays the following message:

```
The set is empty
```

7

Modifying and Maintaining Cubes

Editing a Cube

After a cube is created, it might be necessary to make changes to the cube. Whether you need to change a cube that exists only as metadata or has physical files built, you can modify the cube with the Edit Cube Structure function. This function launches the Cube Designer wizard, where you can make various changes to the cube including the following:

- changing the source data for the cube
- adding or removing structural data to the cube (dimensions, levels, hierarchies)
- changing properties for cube components (level properties, measure attributes, and so on)
- adding or removing cube measures, member properties, or aggregations

The Cube Designer wizard gives you the opportunity to edit and build the cube based on the existing cube metadata. When you are finished making changes to the cube in the Cube Designer wizard, you can choose to update the metadata or update the metadata and build the cube.

To access the Edit Cube Structure function, select the cube that you need to edit in the SAS OLAP Cube Studio tree view. Then select **Edit Cube Structure** from the **Actions** menu or from the cube's context menu. This opens the Cube Designer wizard. The selected cube is opened and can now be edited.

Renaming a Cube Object

It might be necessary at times to rename a cube or other object (such as a job, document, or note) in SAS OLAP Cube Studio. You can change the name of an object by selecting that object in the tree view and then selecting **Rename** from the **Edit** menu or the object's context menu. Enter a new name for the object and click outside the object name. The object is now renamed. For further information about SAS naming conventions see the topic “Naming Guidelines and Rules for the SAS OLAP Server ” on page 51 .

Note: You must have WriteMetadata permissions to rename a cube. For more information about permissions, see the *SAS Intelligence Platform: Security Administration Guide*.

Deleting Cubes and Cube Objects

Deleting a Cube in SAS OLAP Cube Studio

While working in SAS OLAP Cube Studio, you can delete a cube. The Delete function deletes the cube metadata definition and the cube's physical files. If you are deleting a SAS OLAP cube, you should note the difference between the Delete function and the Delete Physical Cube function. The Delete function deletes the entire cube, whereas the Delete Physical function deletes the physical cube files but maintains the cube metadata.

The Delete function should be used on a cube only if you are re-creating the cube from a completely different data set or table. The use of the Delete function will remove all information about a cube, including security information and information maps.

Deleting Cube Objects in SAS OLAP Cube Studio

The tree view in SAS OLAP Cube Studio lists the different objects that are used to create and manage cubes, including tables, libraries, schemas, and jobs. As with cubes, you can delete an object by selecting **Delete** from either the **Edit** menu or the object's context menu. After selecting **Delete**, select **Yes** to delete the object.

Deleting the Physical Cube in SAS OLAP Cube Studio

When building and maintaining SAS OLAP cubes, it might be necessary at times to delete the physical files for a cube but maintain the cube metadata. You can delete just the physical files for a cube by using the Delete Physical Cube function in SAS OLAP Cube Studio. The cube metadata is maintained and the cube is then listed in the tree view as a cube metadata definition.

Using the DELETE and DELETE_PHYSICAL OPTIONS

If you are manually editing and deleting cubes, you can use the DELETE or DELETE_PHYSICAL options with the PROC OLAP statement.

Refreshing Cube Metadata

Overview

When a cube is accessed by a SAS OLAP Server, the cube data is read by the OLAP Server and then reported when queries against the cube are made. It is possible that changes and updates to a cube can occur after the OLAP Server accesses the cube. To make certain the data that is being queried is the most recent, the Refresh Cube function can be used. The Refresh Cube function enables the SAS OLAP Server to access cube data that was created or updated since the cube was last accessed by the SAS OLAP Server. You might also want to refresh a cube that has had changes or functions performed since you opened the cube in SAS OLAP Cube Studio. For example, if a cube has had calculated members or named sets defined by other users, the Refresh Cube function would make the changes visible with the Cube Viewer function.

To refresh a cube, select the cube in the tree view. Select **Refresh Cube** from the **Actions** menu or from the cube's context menu. The Refresh Cubes function then updates the OLAP server metadata for the cube. If the refresh process is successful, you receive a message in the lower left corner of the SAS OLAP Cube Studio window stating: “Cube Refresh of “cubename” is successful”.

You must be connected to a server and have administrative permissions in order to select the Refresh Cubes function.

MDX DDL REFRESH Statement

The REFRESH statement can be sent manually. You can send the REFRESH statement for each additional server that the schema is associated with.

```
refresh cube (cubename | "_ALL_" )
```

Cubename specifies a single cube to refresh for the current server connection. _ALL_ specifies that all cubes are refreshed for the current server connection. Here are some examples.

This example uses the REFRESH statement to refresh the metadata associated with a cube named OrionStar.

```
refresh cube [OrionStar]
```

This example uses the REFRESH statement to refresh the metadata for all cubes managed by the connected server.

```
refresh cube _ALL_
```

You can use the OLAP MDX SQL pass-through facility to send the DLL REFRESH statement to a server. Here is an example.

```
proc sql noerrorstop;
  connect to olap (&olapcon);
  execute
  (
    refresh cube [OrionStar]
  ) by olap;

proc sql noerrorstop;
  connect to olap (&olapcon);
  execute
  (
    refresh cube _ALL_
  ) by olap;
```

Tuning Cube Aggregations

SAS OLAP Cube Studio — Tune Aggregations Function

Overview

After you have built a cube in SAS OLAP Cube Studio, you can add new aggregations to the cube. You can also fine-tune the existing aggregations that were built with the cube and delete any unnecessary aggregations. The Aggregation Tuning function enables you to generate new aggregations based on cube data cardinality or ARM log data. You can also manually build an aggregation, selecting the exact hierarchies and levels that you want to generate aggregations from. To modify a cube's aggregations, select the cube from the tree view and select **Aggregation Tuning** from the **Actions** menu or from the cube's context menu.

Aggregation Tuning dialog box

The Aggregation Tuning dialog box provides functionality to either automatically generate aggregation recommendations or to manually define cube aggregations. Aggregations that are defined with the Aggregation Tuning function are added to a list of any existing aggregations. This list of aggregations can then be reviewed and edited. You can modify performance options for aggregations and, if needed, you can remove any unnecessary aggregations. Finally, you can build the aggregations for the cube. You can also export the resulting PROC code to a text file.

The Aggregations table displays the aggregations for the cube. This includes both existing aggregations and newly created aggregations that can be built. You can select individual aggregations to edit, or you can select multiple, sequential rows of aggregations. The information displayed on the Aggregations table includes the following columns:

Status
: This column indicates the current status of the individual aggregations.

Count
: This is a statistic that is generated from the ARM log. It shows the number of queries that match the aggregation definition.

Total Time
: This is a statistic that is generated from the ARM log. It shows the total wall time (for internally accessing the cube data) for queries that match the aggregation definition. The time value is displayed in hr : min : sec : millisec format.

Average Time
: This is a statistic that is generated from the ARM log. It shows the average wall time (for internally accessing the cube data) for queries that match the aggregation definition. The time value is displayed in hr : min : sec : millisec format.

Additional Options

The Aggregation Tuning dialog box also contains the following options:

Export Code
: This option opens the Export Code dialog box. You can preview and save to a text file the PROC OLAP code that is used to update the cube aggregations. The aggregations that are new, modified, or dropped in the Aggregations table are written to the text file. This button is inactive until you add a new aggregation to the Aggregations table or drop an existing aggregation from the table.

Update Aggregations
: This option enables you to update the aggregations that are listed as new or modified in the Aggregations table. When **Update Aggregations** is selected, the cube is updated to include the new aggregations and changes to any existing aggregations. This button is inactive until you add a new aggregation to the Aggregations table, modify the name of an existing aggregation, or drop an existing aggregation from the table. The Aggregations Tuning dialog box closes after each build.

In the Aggregations Tuning dialog box, you can select one of the following methods to define aggregations for a cube:

ARM Log

If an aggregation already exists, it can have its performance values updated from the Arm log. ARM analysis is used to monitor and diagnose the performance of various SAS applications. It enables you to measure and record application performance and query response times in a designated log file. When using an ARM log to perform ARM analysis of cube aggregations, you can select either of the following options on the ARM :

Create aggregation recommendations and update existing performance values based on the ARM log.
: You can add aggregations from the ARM log and update the statistics for the aggregations that have entries in the ARM log. This is the default option.

Update performance values based on the ARM log.
: You can update the statistics for aggregations that have entries in the ARM log. This option is used when you want to examine the information in the ARM log and verify that the existing aggregations are being used. This can help determine which aggregations to drop.

The **Analyze** button opens the Analysis Recommendations dialog box. From here you can select one or more recommended aggregations. The aggregations that you select are added to the bottom of the Aggregations table as highlighted rows. The **Analyze** button is inactive until text is entered in the **Enter an ARM log file** field.

For ARM log analysis, the generated aggregations list displays all aggregations that can be determined from the cube query data that is provided in the ARM log. If you select all of the generated aggregations and add them to the Aggregations table, you cannot generate further aggregations from that same ARM log. If you reselect the **Analyze** button on the **ARM Log** tab, you receive a message stating: “The ARM analysis did not recommend any additional aggregations to add to the cube”.

Note: For further information about ARM logging, see "SAS OLAP Server Monitoring and Logging" in the *SAS Intelligence Platform: Administration Guide* and the *SAS Interface to Application Response Measurement (ARM): Reference*.

Cardinality

The **Cardinality** tab enables you to add aggregations that are recommended based on the relative cardinality (number of members) of the cube levels. This method of adding aggregations is used when a cube is first created and before an ARM log can be generated. For each aggregation build, the cardinality algorithm generates up to 100 aggregation recommendations that are based on the cardinality ratio between the cube levels. The aggregations with the highest cardinality are recommended. All dimensions are included in the analysis, and any dimension of type TIME has up to the first two of its levels included on each aggregation.

Manual

The **Manual tab** enables you to select the exact hierarchies and levels that you want to generate aggregations from. On the **Manual** tab, the hierarchies for the cube are listed individually as columns. Levels for the hierarchies are numerically listed in drop-down lists on the columns. The default selection value of each column is **None**. When you select an individual level from a hierarchy, you are selecting that level and its parent levels.

Using PROC OLAP to Tune Aggregations

To modify an aggregation through PROC OLAP, use the DROP_AGGREGATION statement to delete the aggregation, and then use the AGGREGATION statement to define the new aggregation.

- DROP_AGGREGATION level-name1level-name2 ...level-nameN > / NAME='aggregation-name' ;
- AGGREGATION level-name1 / < NAME='aggregation-name' > ;

For more information about the DROP_AGGREGATION and AGGREGATION statements, see Chapter 12, “OLAP Procedure,” on page 315 .

Monitoring OLAP Server Performance

SAS OLAP Server performance is monitored and logged with the Application Response Measurement (ARM) interface. The ARM interface provides built-in logging capabilities and generates log records that indicate query content and query start and completion times. From this data, information about aggregation usage, individual query response times, or throughput can be determined. Traditionally, ARM enables system administrators to monitor application executions, run times, performance, and completion. SAS OLAP Server uses ARM to monitor the following:

- application behavior
- user behavior and usage
- server loads
- cube optimization (query response time)
- cube metrics—counts of connections and queries

.

Specifying Tuning and Performance Options in Cube Aggregations

Overview

When you build cubes, you can set various options that improve and optimize cube creation and query performance. These options can be set for all aggregations in a cube or for a specific aggregation. Moreover, these options can be set by using the PROC OLAP options or in SAS OLAP Cube Studio. These options are stored with the cube metadata in the SAS metadata.

Setting Options on the Aggregation Tuning Dialog Box

Overview

In the Cube Designer – Aggregation Tuning dialog box, the **Options** button is provided for access to tuning options. Select the **Options** button to open the Performance Options dialog box. There are two tabs for setting tuning options, the **Default** tab and the **Aggregation** tab.

Default Tab

The default performance options are applied to all aggregations for the cube. These performance options include the following:

- amount of memory (in megabytes) that is available for aggregation creation
- maximum number of threads that are used to create an aggregation index
- number of aggregations to create in parallel
- partition size (in megabytes) of aggregation table partitions
- number of observations (in kilobytes) to include in the index component file segment
- location of index component files
- location of partitions in which to place aggregation table data
- aggregation tables that are stored in compressed format.

For specific information about these functions, see the Performance Options - Default tab Help page in SAS OLAP Cube Studio Help.

Aggregation Tab

The aggregation-specific performance options are applied to an individual aggregation for the cube and override the global option settings for that aggregation. You can define and modify performance options for an aggregation or delete options for an aggregation. The aggregation-specific performance options include the following:

- partition size (in megabytes) of aggregation table partitions
- number of observations (in kilobytes) to include in the index component file
- location of index component files
- location of partitions in which to place aggregation table data

- aggregation tables stored in compressed format
- aggregations created with indexes

For specific information about these functions, see the Performance Options - Define tab Help page in SAS OLAP Cube Studio Help.

Setting Options with PROC OLAP

You can set options for all aggregations in a cube or for a specific aggregation. To set options for all aggregations, set the options in the PROC OLAP statement. To set options for a single aggregation, set the options in the PROC OLAP AGGREGATION statement. The options include the following:

ASYNCINDEXLIMIT= *n*
specifies a limit on the number of indices that will be created in parallel during the cube build process.

COMPRESS | NOCOMPRESS
specifies whether to store the aggregation tables in a compressed format on disk.

CONCURRENT=*n*
specifies the maximum number of aggregations to create in parallel.

DATAPATH=('*pathname1*' ...'*pathnameN*')
specifies the location of one or more partitions in which to place aggregation table data.

INDEXPATH=('*pathname1*' ...'*pathnameN*')
specifies the locations of the index component files that correspond to each aggregation table partition as specified by the DATAPATH= option.

INDEXSORTSIZE=*n*
specifies the amount of memory in megabytes that is available when aggregations are created. The default is the system's available memory.

MAXTHREADS=*n*
specifies the maximum number of threads that are used to asynchronously create the aggregation indexes.

INDEX | NOINDEX
specifies whether to create the aggregations with indexes.

PARTSIZE=*partition-size*
specifies the partition size in megabytes of the aggregation table partitions and their corresponding index components.

SEGSIZE=*rows-per-segment*
specifies the number of observations (table rows) in kilobytes to include in the index component file segment.

WORKPATH=*pathname*
specifies one or more locations for temporary work files.

Note: ASYNCINDEXLIMIT=, CONCURRENT=, INDEXSORTSIZE=, and MAXTHREADS= are available only in the PROC OLAP statement.

For more information about these options, see Chapter 12, “OLAP Procedure,” on page 315 .

Specifying Calculated Members and Measures

Overview

After you have built a cube, you might find it necessary to add members or measures to the cube. You can add calculated members or measures to a cube in the following ways:

- manually by submitting PROC OLAP code with the DEFINE statement.
- using the Calculated Members Wizard

The Calculated Members function enables you to add new members for the Measures dimension. A calculated member is a definition (for a dimension member) that you create and store with the cube. The calculated member value is generated later during query time. You can also define a calculated member as a measure.

Using the Calculated Members Wizard

Overview

The Calculated Members function can be accessed from the SAS OLAP Cube Studio **Actions** menu or from the context menu for a cube. Click **Calculated Members** to open the Calculations dialog box.

In the Calculations dialog box, you can add new calculated members or modify existing members for the selected cube. The list box displays all calculated members that are defined in the metadata for the selected cube. Select **Add** to launch the New Member Wizard and define a new calculated member.

To modify an existing member, select the member and then click **Edit**. The Edit a Calculated Member dialog box appears.

Selecting Calculation Types

When you click **Add** in the Calculations dialog box, the New Member wizard is launched. In this wizard, you select the type of calculation that you want to create. You can select one of the following types:

- **Simple calculations** to create a simple calculation formula. See “Simple Calculations” on page 101 .
- **Time analysis calculations** to create a time-based calculation formula. See “Time Calculations” on page 101 .
- **Custom calculations** to enter a custom calculation formula. See “Custom Calculations” on page 102 .

Simple Calculations

On this page, you select the variables to create a simple calculation. You can select one of the following calculation types:

- Sum
- Difference
- Ratio
- Percent Increase
- Percent Decrease
- Distinct Count

The **Formula** drop-down lists for the members are populated with all measures, including calculated measures. The **Formula** panel changes depending on which calculation-type radio button you select. When a formula is selected, the **Formula** panel is populated with drop-down lists appropriate for the selected calculation. A member must be selected in each list. The member selections for the different formulas are preserved across formulas within a type.

If you select **Distinct Count**, a selection tree is displayed. Each dimension has a node that expands into its hierarchies and then into the levels for each hierarchy. One variable in the hierarchy tree must be selected.

For further information about the Simple Calculations page, see the SAS OLAP Cube Studio Help.

Time Calculations

On this page, you select the variables to create a time calculation. You can select one of the following calculation types:

- Opening Balance
- Closing Balance

- Rolling Total
- Average Over Time
- Compare Parallel Periods
- Compare Consecutive Periods

The **Formula** panel changes depending on the time-calculation radio button that you select. The **Existing measure** drop-down list is populated with all measures, including calculated measures. The **Time period** drop-down list is populated with members from the Time dimension. A member must be selected in each list. The member selections for the different formulas are preserved across formulas within a type.

Note: This option is not available if there are no TIME-type dimensions for the cube.

For further information about the Time Calculations page, see the SAS OLAP Cube Studio Help.

Custom Calculations

On this page, you enter the variables to create a custom calculation. You can enter the following:

Parent dimension
: specifies a parent dimension for the new calculated member that you are creating. The default dimension is Measures. If you select a dimension other than Measures, this field is initially populated with the unique name for the default hierarchy's ALL member.

Parent member
: specifies a parent member for the new calculated member. Use the **Browse** button to display a tree of valid members for the dimension. If you select the **Browse** button and the cube physically exists, you are prompted to log on to an OLAP server, if you haven't already.

  Note: If the cube exists, you must connect to the OLAP server so that the valid members can be retrieved from the server. If the cube does not exist or a connection cannot be made to the OLAP server, the valid members displayed in the tree will be the ALL members for each of the hierarchies for the dimension.

Name
: specifies a name for the custom calculation.

Formula
: specifies an MDX formula for the new calculated member.

Click **Verify** to validate the MDX formula entered in the **Formula** text box. This function is available only for cubes that physically exist. When you select **Verify**, the Log on to an OLAP server dialog box appears. Select a server and enter a user ID and password. A connection is made to the selected OLAP server. Click **Clear** to clear the contents of the **Formula** text box.

The **Generated MDX** text area enables you to view the MDX code that currently exists for the selected cube and any new MDX code that is entered in the **Formula** text box.

For further information about the Custom Calculations page, see the SAS OLAP Cube Studio Help.

OLAP VISUALTOTALS_BEHAVIOR Option

The VISUALTOTALS_BEHAVIOR option enables you to clear total rows, subtotal rows, or both total and subtotal rows. This option can be added to MDX calculated measure definitions wherever they are defined. You can define MDX calculated measures with the PROC OLAP code DEFINE statement and the SAS OLAP Cube Studio Calculated Members wizard.

When VISUALTOTALS_BEHAVIOR is set on a calculated member, and that calculated member is then used in an MDX statement, subtotal and total calculations are not computed. This applies whenever at least one of the listed dimensions or hierarchies is part of the visible axes in the report. Instead, MISSING is returned for the associated total or subtotal member values. The results are blanked out for the total or subtotal result.

The VISUALTOTALS_BEHAVIOR option is valid on the WITH clause in a query, the DEFINE statement in PROC OLAP, and the CREATE SESSION|GLOBAL MEMBER command.

Here is the syntax for the VISUALTOTALS_BEHAVIOR option:

```
VISUALTOTALS_BEHAVIOR|VISUALTOTALS_BEHAVIOR=(BLANK|BLANK_SUBTOTALS|
BLANK_TOTALS, {dimension_name | hierarchy_name}* )
```

Here are the options that you can specify for VISUALTOTALS_BEHAVIOR:

BLANK
specifies that both subtotal and total rows are cleared.

BLANK_SUBTOTALS
specifies that only the subtotal rows are cleared. Total rows are not blanked.

BLANK_TOTALS
specifies that only the total rows are cleared. Subtotal rows are not blanked.

With the VISUALTOTALS_BEHAVIOR option, you can specify one or more dimension or hierarchy names. Here are some items to consider when specifying dimensions or hierarchies with this option:

- Use square brackets [] as part of the name when needed (for example,Time, or [Product Dimension])
- At least one name must be given.
- The dimension/hierarchy list is delimited by curly brackets {}.
- Multiple dimension and hierarchy names are separated by commas.
- Specifying a dimension is equivalent to specifying all hierarchies of that dimension.
- An error is generated at query execution time if any names given do not match the dimensions or hierarchies on the cube.

Here is an example of the VisualTotals_Behavior option:

```
with
member measures.[&1sales_n] as '[measures].[sales_n]',
  visualtotals_behavior=(BLANK, {[DEALERS]})
select
non empty  {[Measures].[SALES_SUM],[Measures].[&1SALES_N] } on 0,
non empty VisualTotals(
AddParents(
  CROSSJOIN([CARS].[CAR].members,
             [DEALERS].[DEALER].members) , POST), "Subtotal *") on 1
```

Here are the results:

```
                                Sum of SALES    &1sales_n
           Chevy       Finch      $10,000.00            1
           Chevy       Smith      $17,000.00            1
   Subtotal Chevy All DEALERS      $27,000.00
         Chrysler       Finch      $40,000.00            2
Subtotal Chrysler All DEALERS      $40,000.00
            Ford       Finch      $30,000.00            3
            Ford       Smith      $56,000.00            4
    Subtotal Ford All DEALERS      $86,000.00
          Toyota       Finch      $15,000.00            1
          Toyota       Jones      $26,000.00            5
          Toyota       Smith      $35,000.00            3
  Subtotal Toyota All DEALERS      $76,000.00
Subtotal All CARS All DEALERS     $229,000.00
```

The results show that for the SALES_SUM measure there are values for all the totals and subtotals. But because the SALES_N measure is using the VISUALTOTALS_BEHAVIOR option to blank out all totals and subtotals, there are no values for it on those rows.

Multiple Language Support for Cubes

Overview

OLAP cube data is often generated in one language and queried in other languages. For example, a company's OLAP cube data might be stored in English, but users who speak Spanish and Turkish need access to it. So, the member values as well as the captions that are assigned to dimensions, hierarchies, levels, and so on, need to be translated. Multiple language support is available only for cubes that are loaded from star schemas.

Multilingual cubes are created with tables of translated dimension data, with one table for each locale. Translated captions are listed in another set of translated tables, again one for each locale. In the OLAP procedure, the multilingual caption tables can be defined for the cube, or for individual dimensions. In SAS OLAP Cube Studio, you create multilingual cubes in the Cube Designer wizard.

Beginning in SAS 9.4, properties for multilingual cubes are automatically updated when reports are generated.

SAS OLAP Cube Studio and Dimension Table Translations

In the Cube Designer – General page, select the **Advanced** button. If you selected **Star Schema** as the input type, you will see the **Dimension Table Translations** tab. From the **Available** list, select the needed languages for the translation tables and move it to the **Selected** list. The first language in the **Selected** list is the default language.

Once you have selected languages for your cube, an additional page will be displayed in the Cube Designer, the Cube Designer – Translated Caption Tables page. On this page, you can indicate the data tables that contain translated captions. Specifically, you can select a single table for the cube or separate tables for separate dimensions. MLSID values connect translated captions to their respective cube elements, such as dimensions and measures.

PROC OLAP Syntax for Multilingual Cubes

The OLAP procedure uses the following language elements to implement multilingual cubes:

USER_DEFINED_TRANSLATIONS statement
specifies a list of the locales that are available for translated dimension tables and caption tables.

Muiltilingual cube options on the PROC OLAP statement:
DTLIBREF specifies a library for translated drill-through tables.

DTMEMPREF specifies the prefix of the translated drill-through table names.

DTMEMPREFOPTS asserts data set options that are applied when you open translated drill-through tables.

CUBETABLELIBREF identifies a library of translated caption tables.

CUBETALBECAPPREF identifies the prefix of the translated caption table names.

MLSID identifies the row in the caption tables that contain the translated cube description. (Cubes do not have captions.)

Note: If captions and descriptions for measures are to be translated, the MLSIDs and related captions must appear in the cube's caption table. In this circumstance, the CUBETABLECAPPREF option is required.

Multilingual options on the DIMENSION statement enable different translations in different dimensions of the cube:
DIMTABLELIBREF specifies a library for translated dimension and/or caption tables.

DIMTABLEMEMPREF specifies the prefix of the translated dimension table names.

DIMTABLECAPPREF identifies the prefix of the translated caption table names.

MLSID identifies the row in the caption tables that contain the translated dimension caption.

Note: The MLSID option is valid in any statement that provides a caption, including the HIERARCHY, LEVEL, PROPERTY, and MEASURE statements.

For more information on PROC OLAP syntax, see Overview: OLAP Procedure on page 315.

MLSIDs specified on the PROC OLAP statement, the MEASURE statement and the DEFINE statement require the use of CUBETABLECAPPREF. Other MLSIDs are associated with the DIMTABLECAPPREF if there is one for that specific dimension. Any MLSIDs that are not associated with a dimension require the use of CUBETABLECAPPREF.

Structure and Naming of Translated Dimension Tables

The languages that are supported by a multilingual cube are specified in the USER_DEFINED_TRANSLATIONS statement. Locale names are specified in 5-character Posix notation. Your multilingual cube can support any of the locales that are valid for the SAS system option LOCALE=, as listed in the *SAS 9.4 National Language Support (NLS): Reference Guide*.

The Posix naming convention for locales is also used in the names of the translated dimension tables. Dimension tables share a common base name. The base name is specified by DIMTABLEMEMPREF. The base name, without a suffix, identifies the dimension table for the default language. The dimension tables for the other languages are named with the base name plus the Posix locale name. For example, if you have a dimension table named CUSTOMERS_, and if English is the default language, then the translated copies of that dimension table could be named CUSTOMERS_de_DE (German), CUSTOMERS_es_AR (Spanish as used in Argentina), and CUSTOMERS_ja_JP (Japanese).

For each translated dimension, you need to create one translated dimension table for each locale that you specify in the USER_DEFINED_TRANSLATIONS statement.

Structure and Naming of Translated Caption Tables

Translated caption tables share the same naming requirements as translated dimension tables. You need at least one translated caption table for each language in your USER_DEFINED_TRANSLATIONS statement.

Within your translated caption tables, captions are referenced by a numeric MLSID. as shown in the following example.

```
The SAS System

Obs    mlsid          caption          description

  1    1              Customer         CustomerDescription
  2    2              Customer         CustomerHierDescription
  3    3              custRegion       CustomerRegionLevelDescription
  4    4              custState        CustomerStateLevelDescription
  5    5              custCity         CustomercityLevelDescription
  6    6              custName         CustomerNamePropertyDescription
  7    7              custGender       CustomerGenderPropertyDescription
  8    8              custEducation    customerEducationPropertyDescription
  9    9              custmvhhcod      customerMVHHCODEPropertyDescription
 10    10             custNkids        customerNKIDSPropertyDescription
 11    11             custMarried      customerMaritalStatusPropertyDescription
```

The preceding example shows the columns that are required in translated caption tables. Any translated copies of the preceding table would apply the same MLSID to the same translated caption. MLSID values can be assigned to any cube object that can receive a caption.

You can structure your caption tables in three ways:

Cube
: For the entire cube, you can use a single set of translated caption tables for all translated dimensions. To implement this scenario, specify a value for the PROC OLAP statement option CUBETABLECAPPREF. Do not specify values for DIMTABLECAPPREF in your dimension statements. Any caption that is not identified by an MLSID will appear in the default language.

Dimension
: To apply translated caption tables within a single dimension, create translated caption tables that apply to that dimension only. Specify a value for DIMTABLECAPPREF and do not specify a value for CUBETABLECAPPREF.

Cube and Dimension
: To combine generic captions with dimension-specific captions, specify both CUBETALBECAPPREF and DIMTABLECAPPREF. MLSIDs on dimensions that have not specified a DIMTABLECAPPREF will refer to observations in the tables specified by CUBETABLECAPPREF.

The following table illustrates the how translated caption tables are defined and named at the cube level (PROC OLAP statement using CUBETABLECAPPREF) and at the dimension level (DIMENSION statement using DIMTABLECAPPREF).

Table 7.1 *Naming Convention for Translated Caption Tables*

Locale	Default	Dimension Data Sets		
English	No	ctimedim_en_US	cardim_en_US	dealdim_en_US
Japanese	No	ctimedim_ja_JP	cardim_ja_JP	dealdim_ja_JP
Polish	Yes	ctimedim_	cardim_	dealdim_

SAS Servers and Character Encoding

If your server metadata contains characters other than those typically found in English, then you must be careful to start your server with an ENCODING= or LOCALE= system option that accommodates those characters. For example, a SAS server started with the default US English locale cannot read metadata that contains Japanese characters. SAS will fail to start and log a message indicating a transcoding failure.

In general, different SAS jobs or servers can run different encodings (such as ASCII/EBCDIC or various Asian DBCS encodings) as long as the encoding that is used by the particular job or server can represent all the characters of the data being processed. In the context of server start up, this requires that you review the characters used in the metadata describing your server (as indicated by the SERVER= object server parameter) to ensure that SAS runs under an encoding that supports those characters.

Single Encoding Required for Cube and All Translated Tables

All of the translated dimension and caption tables for a given multilingual cube must be created with an encoding that applies to all of the translated languages. For example, if a cube supports the English, French and German locales, then the tables can use the Latin1 encoding. A more generic encoding is necessary for a more divergent set of supported languages. For example, a cube that supports the English (Latin1), Polish (Latin2), and Japanese (Shift JIS) locales, the cube and all tables need to use the encoding UTF-8, which is common to all three languages.

If table encodings differ when the cube is built, then all drill-through options are ignored. The cube will be built without drill-through capability.

Adding SAS System Options to a Cube

When you build an OLAP cube, it is often necessary to include additional SAS code that must run prior to the creation of the cube. This includes the creation of user-written formats, PROC statements, and format search paths for the formats that are used on input tables. The Advanced Cube Options dialog box is accessed from the Cube Designer - General dialog box. It provides two entry tabs, **Submit SAS Code** and **Format Search Path**. Both tabs provide entry fields for SAS code. You can enter any text in the fields. There is no validation of the text that is entered. However, error messages are sent to the SAS log.

The text is saved to the cube metadata in the SAS Metadata Repository and is used every time the cube is created. You can edit or remove the text after it is initially

entered. Highlight the text and make any needed changes, or use the DELETE key to remove the text. Select **OK** to save your changes.

Submit SAS Code

You can use this tab to enter a PROC statement or any SAS code that you want to submit before the cube is created. SAS code is submitted before any format search path.

Format Search Path

You can use this tab to enter names of catalogs or libraries for the format search path. The catalogs and libraries must be separated by a blank and will be searched in the order in which they are listed. You use the SAS system option FMTSEARCH= here.

Note: For more information about SAS formats, see “Formats” in SAS Language Reference: Dictionary.

Note: When you build a cube with SAS OLAP Cube Studio, the format search path is saved with the cube metadata in the SAS Metadata Repository and used every time the cube is re-created. However, if you submit PROC OLAP code through a SAS session, outside of SAS OLAP Cube Studio, the format search path is ignored. PROC OLAP does not read the information from the SAS Metadata Repository or write the information to the SAS Metadata Repository.

Synchronizing Column Changes

The Synchronize Column Changes function in SAS OLAP Cube Studio enables you to synchronize a cube when a table for an existing cube has encountered a column name change. This function finds the name differences between the cube and its input table and updates the internal cube files with the name change. The Synchronize Column Changes function is available for a cube if the cube physically exists and you have WriteMetadata permissions for that cube.

You can access the Synchronize Column Changes function in SAS OLAP Cube Studio from the **Actions** menu or from the context menu for a cube. If you are not connected to a workspace server when you select a cube, you are prompted to select a workspace server.

Note: You cannot update an unregistered cube.

Accessing OLAP Cubes from SAS: SQL Pass-Through Facility for OLAP

Overview

The SQL Pass-Through facility for OLAP enables a SAS user to connect to an OLAP server and execute queries against OLAP data within the PROC SQL environment. PROC SQL establishes a connection to an OLAP server by using the PROC SQL CONNECT statement.

After a connection is made to the OLAP server, multiple queries can be submitted by using the OLAP multidimensional expressions (MDX) query language. These queries are run against existing OLAP cubes. While using the SQL Pass-Through facility, you can use the MDX DDL statement to create and delete named sets and calculated members, scoped either globally, or for the current server connection. A PROC SQL query is then closed after all observations (rows) of data are returned. To disconnect from the server, you must submit the PROC SQL DISCONNECT statement.

Conversion Issues

Overview

OLAP cube data is multidimensional and flexible in regard to data name lengths and restrictions. However, when PROC SQL sends a query to the OLAP server, data is returned in a flattened, tabular format that contains rows (observations) and columns (variables).

The SAS OLAP Server has unique naming conventions that specify valid column names, lengths, and types. Column names that are returned from the SAS OLAP Server can contain characters (periods, spaces, brackets) and can be unrestrained in length. Moreover, OLAP data types can be variable-length strings, floating-point numbers, or integers. This differs from SAS data set naming conventions, and some conversion is necessary.

VALIDVARNAME

The SQL Pass-Through facility supports the existing SAS option VALIDVARNAME. You can specify the VALIDVARNAME option to control variable names. The current

default setting for VALIDVARNAME is V7, and variable names can be a maximum of 32 characters in length. Each variable must start with a letter or the underscore character and can contain letters, underscores, and numbers. Uppercase and lowercase letters are also allowed.

When converting column names to SAS variable names, the SQL Pass-Through facility for OLAP will do the following:

- truncate the column name to the maximum size that is allowed
- replace any invalid characters with an underscore
- use a numeric suffix to differentiate between duplicate variable names that are generated during the data conversion

Note: For additional information about naming restrictions for the SAS OLAP Server, see “Naming Guidelines and Rules for the SAS OLAP Server ” on page 51 .

Note: For further information about the VALIDVARNAME= system option, see “VALIDVARNAME=System Option” and “Names in the SAS Language” in the SAS Language Reference: Dictionary.

Data Types

OLAP query results that contain member names or strings are converted to a fixed length CHAR type. All OLAP numeric types are converted to standard SAS numeric types (8-byte floating point). Missing values are handled by standard SAS conventions.

Specifying Member Names

When specifying member names, single quotation marks and double quotation marks are uniquely processed by the SQL Pass-Through facility. The following exceptions occur when using quotation marks in member names.

A member name contains a single quotation mark.
: If a member name in the OLAP query contains a single quotation mark (for example, Adam's), the entire query must be enclosed in double quotation marks when passed to the SQL Pass-Through facility. This ensures that the single quotation mark is maintained in the query text. The double quotation marks are then removed before the OLAP query is processed on the OLAP server.

  Note: Macro variables are not expanded within strings that contain single quotation marks. This includes MDX statements enclosed in single quotation marks.

A member name contains double quotation marks.
: If a member name in the OLAP query contains double quotation marks, the entire query must be enclosed in single quotation marks when passed to the

SQL Pass-Through facility. This ensures that the double quotation marks are maintained in the query text. The single quotation marks are then removed before the OLAP query is processed on the OLAP server.

Here is an example of the SQL Pass-Through facility with quotation marks. The member name Adam's requires the double quotation marks around the SELECT statement.

```
proc sql;
connect to olap (host=localhost port=5451 user=user pass=pass);
         select * from connection to olap (
              "select [A].[All A].[M].[Adam's] on Rows,
                 crossjoin([Measures].defaultMember, [B].defaultMember)
                 on Columns
                   from [QuoteTest]"
       );
disconnect from olap;
quit;
```

SQL Pass-Through Example

In the following example, PROC SQL connects to the SQL Pass-Through facility for OLAP to create a new data set named temp, which contains all the variables that are returned from the multidimensional expression (MDX) defined in the SELECT query. The OLAP server returns query results in a tabular format known as a flattened rowset. The table rows become the observations of the output data set, and the table columns become the variables. After all the rows are returned, PROC SQL closes the query. The server connection is terminated when the program encounters a DISCONNECT statement or when the PROC SQL step ends.

Note: Because the OLAP server does not impose the same restrictions on column names, types, and lengths that SAS imposes on data sets, some conversion might be required.

```
100        proc sql;
101           connect to olap (host=localhost service=olap1);
102           create table temp as select * from connection to olap
103           (
104              select { dealers.dealer.members } on 0,
105                     { [cars].[Car].members,
106                       [cars].[Color].members } on 1
107              from mddbcars
108           );
109           disconnect from olap;
110           quit;
```

PROC SQL Syntax

Syntax Options

CONNECT TO OLAP|OLEDB
: establishes a connection to an OLAP server. This statement is required. Specify OLAP to connect to a SAS OLAP server. Specify OLEDB to connect to all other OLEDB for OLAP-compliant servers. When connecting to an OLAP server, implicit connection is not supported. Therefore the CONNECT TO OLAP statement is required. However, when connecting to an OLEDB for OLAP compliant server, implicit connection is supported. For further information about the CONNECT statement, see “Pass-Through Facility for Relational Databases” in the SAS/ACCESS for Relational Databases: Reference.

DISCONNECT FROM OLAP | OLEDB
: ends the connection to the OLAP server.

EXECUTE (MDX DDL statement) BY OLAP | OLEDB
: specifies an MDX DDL statement for creating or deleting calculated members or named sets. This option is executed by the SAS OLAP Server. For further information, see “Basic MDX DDL Syntax” in the *SAS OLAP Server: MDX Guide*.

SELECT . . . FROM CONNECTION TO OLAP | OLEDB (MDX statement);
: specifies the MDX query that is sent to the connected SAS OLAP server.

Verifying the SQL Connection with the SQLRC Macro

The *SQLRC* macro can be used to ascertain the success or failure status of a PROC SQL connection and its contents. It is important to run the *SQLRC* macro immediately after any statement (or SQL sub-statement) that could potentially cause an error. After PROC SQL is run, the *SQLRC* macro is set to indicate the success or failure status of the operation. A value of 0 means success and any other value indicates a failure.

Connecting Directly to a SAS OLAP Server

When connecting directly to a SAS OLAP server, you specify a SAS OLAP server in the CONNECT statement. For example:

```
CONNECT TO olap (host=localhost service=olap1);
```

You can also submit MDX commands to query SAS OLAP cubes and create various temporary OLAP components that can be used during a PROC SQL session including named sets and drill-through paths.

Here are the connection options for a SAS OLAP Server that is started with bridge protocol:

HOST=*machine-name*
specifies either the DNS name or the IP address of the machine that is hosting the OLAP server.

PORT=*port-number* | SERVICE=*service-name*
either the *port-number* or *service-name* is required. The *port-number* specifies the numeric value of the port on which the OLAP server resides. The *service-name* is used to look up the port number of the machine that is hosting the OLAP server.

USER=*userid*
a string that specifies the user's identification for the specified SAS OLAP server. If included, this option is enclosed with parentheses with the required arguments and any options.

PASS=*password*
a string that specifies the password for the user who is identified with the USER= option. If included, this option is enclosed with parentheses with the required arguments and any options.

Note: For detailed information about PROC SQL syntax, see “Overview of the Pass-Through Facility” in the SAS/ACCESS for Relational Databases: Reference and “Syntax for Remote SQL Pass-Through (RSPT) Facility” in the SAS/SHARE User's Guide.

Connecting to a SAS OLAP Server With a Logical Connection

With the increased implementation of loadbalanced OLAP Servers, there are some advantages to using a logical server connection to allow for PROC SQL passthru to OLAP. In releases prior to SAS 9.4, the connection string for SQL Pass-Through requires a specific host and port defined in the CONNECT statement. A possible problem occurs when the specified host and or port you choose is unavailable and a connection cannot be established. This could be the case during routine maintenance and service. A business would still want to take full advantage of having a loadbalanced OLAP server. If one host is down, the administrator still wants the connection to be made to one of the other available hosts.

This feature allows for the SQL Pass-Through Facility for OLAP to use a logical server. This logical server determines the specific SAS OLAP server to actually connect to. This provides for both fail-over and load balancing between multiple SAS OLAP Servers. A SAS Metadata Server is used to find the best SAS OLAP Server to access. For this mechanism to work, a SAS Metadata Server must be available and it's access parameters defined as SAS Options.

Instead of specifying HOST=, you can now specify SERVER=. The PORT= is also omitted. This allows the metaserver to determine which specific OLAP server is the best one to connect to, based on availability and current load. For this approach to work, the METASERVER, METAPORT, METAUSER and METAPASS SAS options must be properly set.

Here is the syntax for the logical server connection:

```
OPTIONS METASERVER="MyMetaServerHostName" METAPORT=9999
METAUSER="mydomain\metauserid" METAPASS="metapassword";
```

Here are the Metadata Interface options:

METASERVER=
is the network IP (Internet Protocol) address of the computer that hosts the SAS Metadata Server. An example is metaserver=d442.na.sas.com. The maximum length is 256 characters.

METAPORT=
is the TCP port that the SAS Metadata Server is listening to for connections. An example is metaport=5282.

METAUSER=
is the user identification for logging into the SAS Metadata Server. The maximum length is 256 characters.

METAPASS=
is the password that corresponds to the user identification on the SAS Metadata Server. The maximum length is 512 characters

These options may also be set explicitly in an OLAP connection string. To connect to an OLAP server , specify the logical server name using the SERVER= option in the connection string and omit the previously used HOST= and PORT=. Here is an example:

```
proc sql;
  connect to olap (server=mylogicalOLAP user=olapuser pass=olappass);
  select * from connection to olap (select { measures.[sales_n],
measures.[sales_sum] } on 0, cars.members on 1 from mddbcars);
  disconnect from olap;
quit;
```

The METAxxx SAS options may also be given explicitly in the connection string. In this case any existing settings of these options are saved and restored after the PROC SQL executes. Here is an example:

```
proc sql;
  connect to olap (server=mylogicalOLAP user=olapuser pass=olappass
metaserver="MyMetaServerHostName" metaport=9999 metauser="mydomain
\metauserid"
metapass="metapassword");
  select * from connection to olap (select { measures.[sales_n],
measures.[sales_sum] } on 0, cars.members on 1 from mddbcars);
  disconnect from olap;
quit;
```

Depending on the SAS Metadata server configuration, there may be connection delays when some SAS OLAP Servers defined are not running. This can occur when the software needs to wait for a timeout when attempting a connection. This delay averages about 7 seconds per SAS OLAP Server polled. For example, if the Metadata server has 3 SAS OLAP Servers defined for a given logical server (OLAP1, OLAP2 and OLAP3) and OLAP1 and OLAP2 are off-line, it may take around 14 seconds (2*7 seconds) for the connection to be finally made, as connections to the down servers are attempted first.

Connecting to a SAS OLAP Server Using the OLEDB Protocol

When connecting to OLE DB for OLAP data the SAS/ACCESS interface to OLE DB is used. With this approach you specify OLEDB in the CONNECT statement. For example:

```
CONNECT TO oledb (provider=msolap prompt=yes);
```

With this approach, you can create a PROC SQL view of the data or specify MDX statements to access the OLE DB for OLAP data directly. In this approach the MDX statements can be used for read-only access of OLE DB for OLAP data. For further information about the CONNECT statement, see “Syntax for the Pass-Through Facility for Relational Databases” in the SAS/ACCESS for Relational Databases: Reference. Also see “Accessing OLE DB for OLAP Data” in the SAS/ACCESS 9.1 Supplement for OLE DB (SAS/ACCESS for Relational Databases).

Specifying User Authentication Information

When you use the SQL Pass-Through facility to connect to an OLAP server, you can choose how the user authentication information is identified. You can choose to store the user information in the PROC SQL statement or enter the information at the time of connection to an OLAP server.

- **`Enter the user information with the connect statement`**. – The user name and password are included and stored in the PROC SQL syntax. For example:

  ```
  connect to olap (host=localhost port=5451 user="user" pass="pass");
  ```

- **`Enter the user information at the time of connection to an OLAP server.`** – Authentication with an OLAP server occurs at the time of connection and is accomplished with Integrated Windows authentication. You do not have to store a user name or password in the connection string. For example:

  ```
  connect to olap (host=localhost port=5451);
  ```

  This option works in two instances:

  1. If both the OLAP server and the SAS client are started with the -SSPI option, then authentication is attempted using Integrated Windows authentication. This requires that both the OLAP server and the user's computer be Windows PCs that have access to a shared domain server. This usually occurs when both machines are on the same network.
  2. If the –SSPI option is not specified on either the client or the OLAP server, or if the Integrated Windows authentication fails, an attempt is then made to contact a metadata server to create a one-time password. This is possible when the OLAP server and the client use the same metadata server. The access information for the metadata server can either pre-exist in a SAS profile or otherwise be queried using a dialog box at runtime.

Note: For more information about Integrated Windows authentication, see the *SAS Intelligence Platform: Security Administration Guide*.

Impact Analysis and Reverse Impact Analysis

The Impact Analysis function enables you to view the relationship of a cube to the objects that it is associated with. Impact analysis shows the potential impact of changes that you might make to a cube. In SAS OLAP Cube Studio, you can perform impact analysis only on a cube. If a cube has an information map, it is also displayed. The Reverse Impact Analysis function enables you to see how changes to the cube's associated objects could affect the cube. This view shows all tables that the cube consists of, including the input, detail, and aggregation tables. Impact Analysis and Reverse Impact Analysis display only the objects for which you have ReadMetadata permission.

To perform impact analysis on a cube, select a cube in the tree view and select **Impact Analysis** from the **Tools** menu or from the cube's context menu. The Impact Analysis dialog box appears. The dialog box contains the **Impact Analysis** tab and the **Reverse Impact Analysis** tab. Each tab contains two options that enable you to toggle between a tree view and a diagram view of the cube and its objects. The diagram view shows a process flow diagram of the cube.

Each node of the cube analysis also contains a context menu that is available on both the **Impact Analysis** and **Reverse Impact Analysis** tabs. This context menu enables you to see the object's properties, analyze columns, and open (view the data in) a table. You can also print the analysis view at any time.

Note: You can use SAS Data Integration Studio to perform impact analysis on other objects, such as tables and jobs. For further information about the use of the Impact Analysis function in SAS Data Integration Studio, see "Using Impact Analysis" in the *SAS Data Integration Studio: User's Guide*.

Disabling and Enabling Cubes

Disabling Cubes

In SAS OLAP Cube Studio, you can disable and enable cubes to perform various cube changes and adjustments. You can disable a SAS OLAP cube if you need to

perform administrative tasks or make edits to the cube. You might disable a cube to perform the following tasks:

- tuning cube aggregations
- adding calculated members to a cube
- changing the security settings for a cube
- renaming a cube
- editing and rebuilding a cube
- changing the OLAP schema for a cube

Disabling a cube removes it from production. As a result, queries against the cube are affected. Queries that are currently running against the cube are completed. However, new queries against the cube are not accepted by the OLAP server. To disable a cube in SAS OLAP Cube Studio, select a cube from the tree view. Then select **Disable** from either the **Actions** menu or the cube's context menu. The cube is then disabled in the tree view.

Note: You must have Administer permissions on the OLAP server to disable a cube. For more information about permissions, see the *SAS Intelligence Platform: Security Administration Guide*.

Enabling Cubes

You can enable a SAS OLAP cube that you have previously disabled. A cube is disabled in order make edits to the cube or to perform administrative tasks on the cube. Enabling a cube brings it into production, so that new queries against the cube are accepted by the OLAP Server. In order to enable a cube in SAS OLAP Cube Studio, select a disabled cube from the tree view. Then select **Enable** from either the **Actions** menu or the cube's context menu. The cube is then enabled and listed as an enabled cube in the tree view.

After the cube is enabled, you can resume queries to the cube that require SAS OLAP Server sessions. In SAS OLAP Cube Studio, you can also use the View Cube function.

Note: You must have Administer permissions on the OLAP server to enable a cube. For more information about permissions, see the *SAS Intelligence Platform: Security Administration Guide*.

Securing Cubes

About OLAP Member-Level Permissions

OLAP member-level permissions enable you to limit access to SAS OLAP data using filters. Each filter consists of an MDX expression that subsets the data in a dimension as appropriate for a particular user or group. The filters are stored in the metadata as permission conditions. This feature relies on the SAS OLAP Server for enforcement. At query time, the server performs the filtering to determine which dimension members should be returned to the user that submitted the query. This ensures that the filters are applied every time the data is accessed.

When you use OLAP member-level permissions, it is essential to understand these points:

- With OLAP data, permission conditions can be specified only on dimension objects.
- The SECURITY_SUBSET option can affect results. See “About the SECURITY_SUBSET Option” on page 121.
- The members that are returned by the MDX expression must all belong to the dimension on which the permission condition is defined. The returned set of members cannot be a union of members from other dimensions.
- A permission condition that filters a non-default hierarchy must include at least one member of the default hierarchy. If a requesting user does not have access to any members in the default hierarchy, then the query fails with a permissions error.

Assign OLAP Member-Level Permissions

Follow these steps to assign a permission condition to one or more members in a given dimension, for a selected user or group.

> **TIP** If you want to use the same dimension in multiple cubes, create a shared dimension. See “Developing and Managing Shared Dimensions” on page 370.

1. In SAS Management Console or SAS OLAP Cube Studio, right-click the dimension for which you are defining a member-level permission and select **Properties**,
2. In the Properties dialog box, click the **Authorization** tab.

3 In the **Authorization** tab, select (or add) the user or group whose access you want to limit.

4 Use the permissions list to select permissions for the user or group.

5 Click the **Add Authorization** button.

Note: If the **Edit Authorization** button is displayed, a condition already exists for the selected user or group.

6 Click the **Edit Authorization** button to create an MDX filter.

7 Use the Build Formula dialog box to create the MDX filter. To subset users, add identity-driven properties to your filter, as described in the next topic. Identity-driven properties are available on the **Data Sources** tab.

See also the example "Setting Member–Level Permissions" on page 244 .

Identity-Driven Properties

Use the following identity-driven properties in your MDX filters to subset the users that receive member-level permissions.

SAS.ExternalIdentity
: This property translates to optional, site-specific values such as Employee ID. Those values are not automatically stored in the metadata repository and need to be loaded and maintained.

SAS.IdentityGroupName
: This property resolves to the name of the requesting group identity (for example, Portal Admins Group).

SAS.PersonName
: This property resolves to the name of the requesting user identity (for example, SAS Demo User).

SAS.IdentityName
: This property returns the name of either the requesting group identity or the requesting user identity, depending on whether the user ID is a group login or a personal login.

SAS.Userid
: This property translates to the authenticated user ID, normalized to one of the uppercase formats USERID or USERID@DOMAIN (for example, SASDEMO@LXXXXX).

SAS.IdentityGroups
: This property resolves to the names of the groups of which a user is a member.

See also the example "Setting Identity-Driven Security" on page 250.

About the SECURITY_SUBSET Option

After you have set the necessary permission conditions for dimension members, you must indicate whether the OLAP server includes these permission conditions when processing cube queries. In order for a cube to control the roll-up values for designated members, the SECURITY_SUBSET PROC OLAP option must be set to YES when the cube is built. The SECURITY_SUBSET option is used in conjunction with permission conditions to specify whether the roll-up values include; only the members in the permission condition, or all of the members in the cube. If you set SECURITY_SUBSET=YES, then the cell values are recalculated at query time based on the security subset defined by the active permission conditions for the given user. If you set SECURITY_SUBSET=NO, then the OLAP server does not recalculate the cell values. The default value (NO) includes all members within a total.

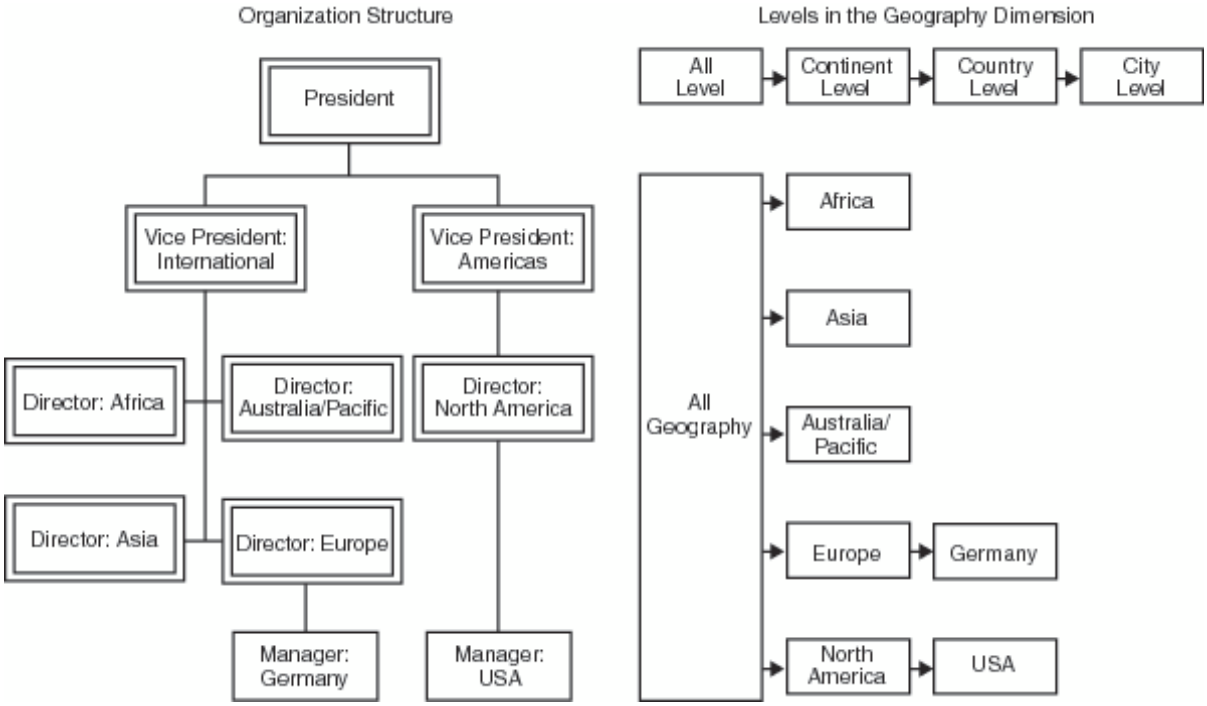
Note: For more information, see the SECURITY_SUBSET PROC OLAP option and the topic "SAS OLAP Security Totals and Permission Conditions" in the *SAS OLAP Server: MDX Guide*.

Example: Using Member-Level Permissions

This example demonstrates how you can use member-level permissions. These are the security goals in this example:

- enable company executives to access data based on their areas of responsibility
- prevent other employees from accessing data in the cube that is used to generate executive reports

The following figure depicts the relevant parts of a company's organization structure and of the OLAP cube against which the company runs executive reports. Notice that the levels in the cube's Geography dimension closely correspond to the depicted organizational structure.



Here are the implementation steps for this example:

1 In SAS Management Console or SAS OLAP Cube Studio, set the following permissions on the cube:

- Give the PUBLIC group explicit denials of the Read and ReadMetadata permissions for the entire cube. The data in the cube is used for executive reports only.
- Give the company president explicit grants of the Read and ReadMetadata permissions for the entire cube. This user sees all of the data.
- Give the SAS System Services group an explicit grant of the ReadMetadata permission. You must always preserve the SAS Trusted User's access to cubes and schemas.
- In most cases, some members of the information technology staff will also need access to the data for administrative purposes. Make sure that any such groups or users have explicit grants of the Read and ReadMetadata permissions.

2 To assign the permission conditions:

a Right-click the cube's **Geography** dimension, and select **Properties**.

b On the **Authorization** tab, select (or add) the Vice President Americas user.

c In the permissions list, add an explicit grant of the Read permission for the Vice President Americas user. This enables the **Add Authorization** button.

d Click the **Add Authorization** button.

e In the Add Authorization dialog box, select the **Create an advanced MDX expression** radio button and click **Build Formula**.

f In the Build Formula dialog box, paste or build a filter for the Vice President Americas user and click **OK**.

g On the **Authorization** tab, click **OK** to save this permission condition and then return to step 2b to repeat the process for the next executive.

For example, the following table contains MDX expressions that you could use to subset the data based on each executive's area of responsibility.

Table 7.2 *Example: MDX Expressions*

User	MDX Expression	Notes
Vice President Americas	{[Geography].[All Geography], Descendants([Geography].[All Geography].[North America],[Geography].[Continent],SELF_AND_AFTER)}	
Vice President International	Except({[Geography].Members}, {Descendants([Geography].[All Geography].[North America],[Geography].[Continent],SELF_AND_AFTER)})	Use Except to exclude North America.
Director North America	{[Geography].[All Geography],[Geography].[All Geography].[North America],[Geography].[All Geography].[North America].Children}	Use .Children to include countries.
	{[Geography].[All Geography],[Geography].[All Geography].[North America],descendants([Geography].[All Geography].[North America])}	Alternative: use Descendants to include countries and cities.
	{[Geography].[All Geography].[North America], [Geography].[All Geography].[North America].Children}	Alternative: exclude the All level.
Director Africa	{[Geography].[All Geography].[Africa], [Geography].[All Geography].[Africa].Children}	
Director Asia	{[Geography].[All Geography].[Asia],[Geography].[All Geography].[Asia].Children}	
Director Australia/ Pacific	{[Geography].[All Geography].[Australia/Pacific], [Geography].[All Geography]. [Australia/Pacific].Children}	
Director Europe	{[Geography].[All Geography].[Europe], [Geography].[All Geography].[Europe].Children}	
Manager Germany	{[Geography].[All Geography].[Europe].[Germany], descendants([Geography].[All Geography].[Europe].[Germany])}	Exclude Europe.[1]
Manager USA	{[Geography].[All Geography].[North America].[USA], descendants([Geography].[All Geography].[North America].[USA])}	Exclude North America.[1]

1 Because this expression excludes a parent level, you should also deny this user the ReadMetadata permission for the level that you are hiding. This is a requirement when OLAP data is accessed through an information map.

Integrated Windows Authentication and Single Sign-On

Single Sign-On (SSO) to SAS IOM Bridge servers

Single sign-on (SSO) is an authentication model that enables users to access a variety of computing resources without being repeatedly prompted for their user IDs and passwords. This functionality enables a client application to connect to a SAS IOM Bridge Server as the currently logged-on user. For example, single sign-on can enable a user to access SAS servers that run on different platforms without interactively providing the user's ID and password for each platform. Single sign-on can also enable someone who is using one application to launch other applications based on the authentication that was performed when the user was initially logged on to the system.

SSO and SAS OLAP Cube Studio

When you have logged on to a metadata server in SAS OLAP Cube Studio, you can perform a variety of functions, including building a cube and saving the metadata for that cube, editing cube metadata, and viewing cube properties. Some functions within SAS OLAP Cube Studio also require a workspace server or an OLAP server in addition to the metadata server. These functions include Build Physical Cube, Incremental Cube Update, Calculated Members, and Cube View. Single sign-on enables you to seamlessly access these servers without having to provide your login credentials. Two technologies are available in SAS 9.4 to accomplish this:

1. Integrated Windows authentication is available for servers that run on either Windows or UNIX.
2. a trusted peer connection is available otherwise.

Note: Before using single sign-on, you must have a user defined in the metadata repository for any user ID that you want to connect to the metadata server with.

Integrated Windows Authentication

A client and server engaged in Integrated Windows authentication use Microsoft's Security Support Provider Interface (SSPI) to choose the best security package for their configuration. When you first open SAS OLAP Cube Studio, you are prompted to log on to a metadata server with a connection profile that you have previously

created. This connection profile contains information about the metadata server, including the machine ID, port ID, user ID, and password. There is also a check box labeled **Use Integrated Windows authentication (single sign-on)**. If this check box is selected, then you will use Integrated Windows authentication (IWA) to connect to the servers (metadata server, workspace server, and OLAP server). You will not need to provide a user ID or password when you open SAS OLAP Cube Studio. The user ID and password of your current Windows session will be used. By default this option is not selected when you create a connection profile in SAS OLAP Cube Studio.

Note: For more information about Integrated Windows authentication see "Integrated Windows Authentication" in the *SAS Intelligence Platform: Security Administration Guide*.

Trusted Peer Connection

The other authentication technology is a trusted peer connection. This method depends heavily on the integrity of the client environment. The IOM Bridge protocol client obtains the identity of the user of the client application from the system environment. When a trusted peer connection is being used, you are prompted during an initial login for your user ID and password when connecting to a metadata server. This is the behavior of previous versions of SAS OLAP Cube Studio. However, when a connection to a workspace server (submitting procedure code) or a connection to an OLAP server (cube viewer) is made in SAS OLAP Cube Studio, you are no longer prompted to log on. The connection is created using the metadata server login credentials.

Note: For more information about trusted peer connections, see “Trusted Peer” in the *SAS Intelligence Platform: Security Administration Guide*.

Exporting Code

In SAS OLAP Cube Studio, you can export PROC OLAP code that you have created while using various functions and wizards, including Cube Designer, Aggregations Tuning, and Incremental Update. The Export Code function enables you to save PROC OLAP code to a designated file. When using the Export Code function, some of the metadata connection profile information is saved to the file. The host and port information for the metadata server is saved to the exported file. In addition, the information is displayed on the Finish page of the different wizards. However, the user ID and password are not saved to the exported file or displayed on the Finish page. This is because single sign-on authentication removes the need to display or save the user ID and password.

Security for Drill-through Tables

Different users of cube data might have different security and access restrictions that must be complied with and applied when querying the underlying data for a cube. When selecting a data table for drill-through, you might need to define user restrictions for certain data in the drill-through table. The SAS Metadata LIBNAME engine is used to assign the drill-through table library on a per-session basis. This allows client credentials to be used when determining which columns the user can see. Columns which have ReadMetadata permissions denied on the drill-through table will not be visible to the user. If access is denied for a column on the drill-through table, that level must also be denied Read access in the cube. Conversely, if a level is denied Read access in the cube, that column in the drill-through table must have ReadMetadata access denied. For more information about the SAS Metadata LIBNAME engine see the topic “Pre-assigning Libraries” in the *SAS Intelligence Platform: System Administration Guide*.

Because of this enforcement, the following changes could affect your site:

- User-defined formats will not work if the FORMATS catalog is accessed by the same libref as the data. You must specify a different libref for the FORMATS catalog or move the user-defined formats into a different location and assign a new library.
- If SAS Trusted User does not have ReadMetatdata permissions on the library definition, then the drill-down functionality fails. You can grant ReadMetadata permission to the SAS Trusted User by using SAS Management Console.
- You cannot drill down in an OLAP cube if there are discrepancies between the physical table and the metadata that is defined for the table. You can update the metadata for the table by using the Update Metadata function in SAS Management Console or SAS Data Integration Studio. If the table is part of a job in SAS Data Integration Studio, then check the code for the job to verify that there are no LENGTH statements that would cause a mismatch to occur.

Note: For further information about drill-through tables, see “Defining Drill-Through Tables” on page 64 .

Applying Batch Security with Permission Tables

When applying permissions to a cube, you might need to address permissions for different combinations of users, groups, SAS OLAP Servers, schemas, and cubes, as well as different elements of the cube, including the dimensions, hierarchies, levels, and measures. For example, you might need to grant ReadMetadata and Read access to the group that contains specific cube users. Or you might need to restrict Read access for different components of a cube (dimension, hierarchy, level, or measure) using MDX conditions for each cube component, per user, consumer, or group. These various combined permission settings can be easily created and managed with batch security that is applied through permission tables.

Starting with SAS 9.2M3, you can specify batch security in SAS OLAP Cube Studio and SAS Data Integration Studio with the Manage Permission Tables function. The Manage Permission Tables function enables you to create a special SAS data set known as a permission table that contains cube access controls for submitting in bulk. A permission table is a table of access control information that can later be applied to a cube with batch SAS code. The Manage Permission Tables dialog box enables you to create and modify permission tables as well as import access controls (permissions) from a cube or an OLAP schema. You can also execute the code interactively or export the code to a file for use in a stored process or deployed job flow.

When a cube is created, security for that cube is determined by the permission settings that are found in the cube metadata. In SAS OLAP Cube Studio, permission tables will appear in the metadata tree as a table. You must have WriteMetadata access to create and modify permission tables. To access the Manage Permission Tables function in SAS OLAP Cube Studio, select **Tools** ⇨ **Manage Permission Tables**. The Manage Permission Tables dialog box appears.

Permissions tables have the following columns.

Table 7.3 *Columns in Permissions Tables*

Variable Name	Type	Length	Description
FULLNAME	Char	32	The OMR identity (person or group).
OLAPSCHEMA	Char	32	The name of the OLAP schema for the cube.
CUBE	Char	32	The name of the cube.
DIMENSION	Char	32	The dimension name or a blank value if an ACE on measures is required.
ITEMS	Char	32	A list of names of hierarchies, levels, or measures, separated by blank spaces.
PERMISSION	Char	32	The access permission, such as Read and ReadMetadata.
PERM_TYPE	Char	2	The code that identifies the type of the ace. See the following table for type definitions.
MDX_CONDITION	Char	32,000	The MDX string that contains the condition. This value is blank if no condition must be provided, or if a previous MDX condition has to be removed.
REMOVE_ACE	Char	1	A flag that specifies if the ACE is to be added or simply to be removed from the OMR.

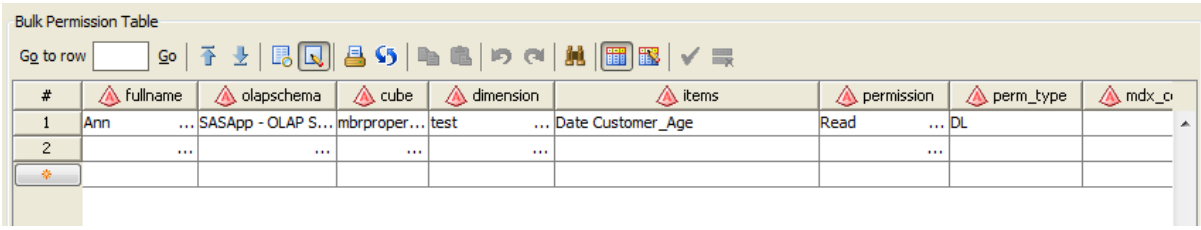
Table 7.4 Codes in the PERM_TYPE Column of Permissions Tables

PERM_TYPE	Short Description	Description
DC	Deny Cube	Denies ReadMetadata' permission on the specified cube. Other columns are ignored.
GC	Grant Cube	Grants ReadMetadata permission on the specified cube. Other columns are ignored.
DD	Deny Dimension	Denies ReadMetadata permission on the specified dimension. Dimension field must be specified. Other columns are ignored.
GD	Grant Dimension	Grants ReadMetadata permission on the specified dimension. Dimension field must be specified. Other columns are ignored.
DH	Deny Hierarchy	Denies ReadMetadata permission on the specified hierarchy. Dimension field must be specified. ITEMS column must contain the name of the hierarchy. Other columns are ignored.
GH	Grant Hierarchy	Grants ReadMetadata permission on the specified hierarchy. Dimension field must be specified. ITEMS column must contain the name of the hierarchy. Other columns are ignored.
DM	Deny Measures	Denies ReadmetaData permission on the specified measure. Dimension field must contain `Measures` value. ITEMS column must contain the name of the measure to deny.
GM	Grant Measures	Grants ReadmetaData permission on the specified measure. Dimension field must contain `Measures` value. ITEMS column must contain the name of the measure to deny.
DL	Deny Levels	Denies ReadMetadata permission on the specified levels. Dimension field must be specified. ITEMS column must contain the name of the levels to deny. If more than one, then they must be separated by a blank. If an MDX condition is specified, then it sets a corresponding ACI for the dimension.
GL	Grant Levels	Grants ReadMetadata permission on the specified levels. Dimension field must be specified. ITEMS column must contain the name of the levels to deny. If more than one, then they must be separated by a blank. If an

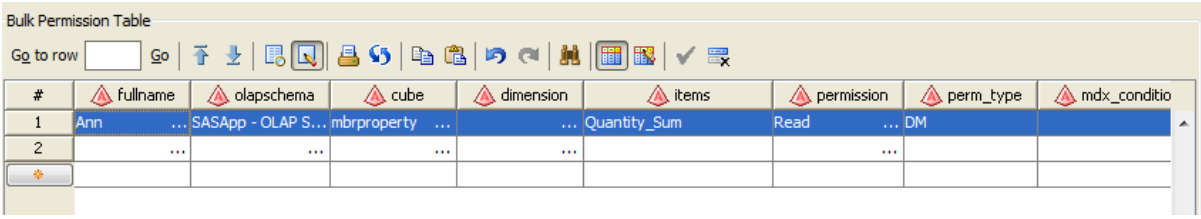
PERM_TYPE	Short Description	Description
		MDX condition is specified, then it sets a corresponding ACI for the dimension.
<blank>		Sets the ACI with the provided MDX condition If the MDX condition is empty, then it removes the condition (but not the ACE) for that dimension.

Permissions Table Examples

The following display shows the denial of Read access to the levels Date and Customer_Age. Note the blank-delimited entry in the Items column.



The following display shows the denial of Read access for the measures Quantity and Sum.



8

Updating SAS OLAP Cubes

Overview

After a SAS OLAP cube is created, it might be necessary to add further data to the cube without completely re-creating the cube. If a cube is in production, you must consider what the best process to update your cube is. You should also consider an approach that will have as little impact as possible on users. It is important to limit the amount of time that a cube is unavailable to users while the cube is being updated with new data. It is also important that the updated cube be checked for correctness before the cube is made available for queries. To address these concerns, you can choose either to update a SAS OLAP cube in-place or to create an incremental copy of the cube.

You can update SAS OLAP cubes that physically exist and that are generated from either detail tables or star schemas. You cannot update a fully summarized cube. Furthermore you must have ReadMetadata and WriteMetadata permissions for all elements of the cube

New data to add to a cube can include new members and new data for the cube or additional data for existing members. You can update a cube using SAS OLAP Cube Studio or manually using PROC OLAP and PROC OLAPOPERATE.

Updating a Cube In-Place

Depending on the business needs for a cube, it might be more suitable to update a cube in-place. A separate version of the cube is not created. The process of updating in-place enables you to add data to a cube while that cube is still online. The cube maintains its name and OLAP schema assignment. This process might be more appropriate for some users and their reporting needs, as it is less complicated than creating incremental generations of a cube. However, there is no opportunity to test and verify the update in a separate process.

You can update a cube in-place by using the UPDATE_INPLACE PROC OLAP option. This option is used with the ADD_DATA option to signal that an update will occur. Here is an example of the UPDATE_INPLACE option:

```
proc olap data=mylib.newdata cube=cubeA add_data
```

```
   update_inplace;
metasvr host="myhost" port=8561 protocol=BRIDGE repository=my repository
   olap_schema=prodSchema;
run;
```

The process of updating in-place does not create a new metadata registration when the update is complete. The existing metadata registration points to the new cube generation and the old generation is deleted. In addition, you do not have to disable the cube prior to beginning the in-place update. OLAP server sessions referencing the old cube will continue to do so until the last session closes the old cube. After that, new sessions will begin to reference the updated cube. However, it might be desirable to have a specific “cross-over” to the new cube at some point after the update completes.

See “Update a Cube In-Place” on page 223 for a discussion of updating a cube in-place with the SAS OLAP Cube Studio Incremental Update Wizard.

Incremental Updates of Cubes and Cube Generations

Overview

When you create a SAS OLAP cube, data from a table is summarized and sub-aggregates are created for the cube. The Incremental Update function enables you to create an incremental update or copy of the original cube. This is known as a generation of the cube. A generation is a complete, independent version of the original cube, with a separate metadata registration and separate sets of member and property trees. The new generation shares the original data (previous aggregation data) rather than maintaining duplicate copies.

The new cube generation enables the original cube to remain online for queries during the entire update process. This is because the original cube is not modified. The original cube continues in production as before while the new cube is available for administrative review and updates (such as security updates or new global calculated members). This provides the administrator with an opportunity to examine and verify the correctness of the updates before bringing the cube online for production without affecting the original cube. A new cube generation also provides the additional benefit of keeping historical data (in the form of the previous generations) available for as long as necessary.

You can update a cube incrementally by using the OUTCUBE or the OUTSCHEMA options. These options are used with the ADD_DATA option to signal that an update will occur and a new generation of the cube will be created. You can use either option separately or both together.

OUTCUBE Option

Here is an example of the OUTCUBE option. In this example, the cube PRODSCHEMA is updated and a cube generation (TSTCUBE) is created in the same schema as the original cube.

```
proc olap fact=olapsio.factcars cube=PRODCUBE
              OUTCUBE=TSTCUBE
              ADD_DATA;
    metasvr host=&host port=&port protocol=&protocol userid=&userid pw=&pw
    repository=&repos olap_schema=PRODSCHEMA;
run;
```

OUTSCHEMA Option

You can also create a generation of a cube by specifying a different schema than the original cube. In this example, the new cube retains the name of the original cube PRODCUBE (OUTCUBE was not specified). However, the new cube now resides in the schema TESTSCHEMA, as specified in the OUTSCHEMA option.

```
proc olap fact=olapsio.factcars cube=PRODCUBE
             OUTSCHEMA=TESTSCHEMA
             ADD_DATA;
    metasvr host=&host port=&port protocol=&protocol userid=&userid pw=&pw
    repository=&repos olap_schema=PRODSCHEMA;
run;
```

OUTCUBE and OUTSCHEMA Options

You can use both OUTCUBE and OUTSCHEMA options to create a new cube in a different schema. In this example, a new cube called TSTCUBE is created and it resides in the schema TESTSCHEM.

```
proc olap fact=olapsio.factcars cube=PRODCUBE
              OUTSCHEMA=TESTSCHEMA
              OUTCUBE=TSTCUBE
              ADD_DATA;
    metasvr host=&host port=&port protocol=&protocol userid=&userid pw=&pw
    repository=&repos olap_schema=PRODSCHEMA;
run;
```

The possible functions that you can perform on a SAS OLAP cube depend on whether the cube is the original cube or a generation of a cube. These functions can also depend on which generation of the original cube it is. Any generation of a cube can have calculated members added to or deleted from it. However, only the most current cube generation can be edited, rebuilt, tuned, or updated. Furthermore, you must have ReadMetadata and WriteMetadata permissions for all elements of the

cube. See “Generating a New Cube” on page 227 for a discussion of creating a cube generation with the SAS OLAP Cube Studio Incremental Update Wizard.

Note: The SAS OLAP Cube Studio right-click menu options for Edit, Create, Manual Tuning, Advanced Tuning, and Delete Physical Cube are not active if the cube is not the current generation.

Coalescing Cube Aggregations

The Incremental Update function enables you to add new members to a cube and update data. Every time a cube update is done, the incremental aggregation data is written to a new SPDE table (rack). The more cube updates that have been completed, the more tables or racks there are that must be read during a query, and the slower the access time can be.

Periodically, it can be beneficial for performance reasons to combine all of the individual tables into a single aggregation table. Aggregations are coalesced (combined) in a separate job from adding data to a cube. Only aggregations with more than one rack are coalesced. The COALESCE_AGGREGATIONS option can be executed only for the most current generation of the cube. In addition, only MOLAP aggregations (those aggregations created by PROC OLAP) can be coalesced. The coalesce task is processed in-place. A new generation of the cube is not created.

After the cube generation is coalesced, it is no longer dependent on the aggregation data from previous generations. If those generations have already been deleted, then the old aggregation tables are also deleted.

You can coalesce all the aggregations by using the COALESCE_AGGREGATIONS option in the PROC OLAP statement. In addition, the Coalesce wizard is available in SAS OLAP Cube Studio.

See “Coalesce Incremental Data for a Cube” on page 231 for a discussion of coalescing cube aggregations with the SAS OLAP Cube Studio Incremental Update wizard.

Updating a Cube in a Production Environment

Overview

When a new generation of a cube is created, it can be designated as the production version of the original cube. The new generation must be given a different name

from the original cube or reside in a different OLAP schema. In order to update a cube in production, you must disable the existing production cube, rename the cubes, and then enable the new production cube.

Disable the Production Cube

To disable a cube, you can use either the SAS OLAP Monitor plug-in to SAS Management Console or the PROC OLAPOPERATE statement. When a cube is disabled, it is taken offline and is not available for new queries. All existing query results are closed, but the sessions are left open and can be closed by the SAS OLAP Server Monitor. If a query is being processed, however, the disable action will fail.

Rename the Cubes

Rename the original cube to another name (for example, Sales_OLD). Then rename the cube generation to the original cube name (Sales). To rename a cube, you can use either SAS OLAP Cube Studio or the PROC OLAP RENAME option. The RENAME option enables you to reassign cube generations. It is used with the OUTCUBE or the OUTSCHEMA options. Renaming a cube updates the metadata for the cube but does not change the file structure or physical location of the cube. To rename a cube, you must have ReadMetadata and WriteMetadata permissions on all parts of the cube and the cube must be disabled.

Enable the New Production Cube

To enable a cube, you can use either the SAS OLAP Monitor plug-in to SAS Management Console or the PROC OLAPOPERATE statement. Enabling the cube enables queries to resume on the server for the new production cube. Existing reports will continue to work as they reference the cube by name.

Example 1

Here is an example where an existing production cube is replaced with an updated version in the same schema. The old cube is then deleted.

Perform an incremental update on production cube A.

```
proc olap cube=A outcube=A_New add_data data=lib1.data1;
metasvr .... ;
run;
```

Disable the production cube.

```
proc olapoperate <server-connection-options>;
disable cube A;
```

```
run;
```

Rename the production cube to a temporary name.

```
 proc olap rename cube=A outcube=A_Old;
metasvr ...;
run;
```

Rename the new cube to the original production cube name.

```
proc olap rename cube=A_New outcube=A;
metasvr ...;
run;
```

Enable the new production cube (make available for queries).

```
proc olapoperate <server-connection-options>;
enable cube A;
run;
```

Delete the old cube.

```
proc olap cube=A_Old delete;
metasvr ...;
run;
```

Example 2

In this second example, an existing production cube is replaced with an updated version from another schema and the old cube is archived.

Perform an incremental update on production cube A. This creates a cube A located in OLAP Schema TestSchema.

```
proc olap cube=A outschema=TestSchema add_data data=lib1.data1;
metasvr .... olap_schema=ProductionSchema ;
run;
```

Disable the production cube.

```
 proc olapoperate <server-connection-options>;
disable cube A;
run;
```

Move the old cube to an archive schema.

```
proc olap rename cube=A outcube=A_September2005 outschema=ArchiveSchema;
metasvr ... olap_schema=ProductionSchema;
run;
```

Move the new cube to the original production schema.

```
 proc olap rename cube=A outschema=ProductionSchema;
metasvr ...olap_schema=TestSchema;
run;
```

Enable the new production cube (make available for queries).

```
 proc olapoperate <server-connection-options>;
enable cube A;
run;
```

Archiving and Deleting Cube Generations

If multiple generations of a cube have been created, special considerations are needed when archiving or deleting these generations. It is possible to archive an old cube generation into another schema or under another name. The cube will then be available for access to historical data. If you are deleting cube generations, it is important to note that a cube that is an earlier generation must be deleted with PROC OLAP. Deleting only the metadata will not work for generations of cubes.

Because cube generations generally look through to aggregation data in previous generations, cube data cannot be physically deleted as long as another generation is dependent upon it. Only those physical files that are not needed by the new cube (for example, member and property trees) are actually deleted. The aggregation files remain available to subsequent generations. In addition, the metadata registration in the metadata repository is deleted.

Updating the Captions and Descriptions for a Cube

You can update the captions and descriptions for a cube without adding data to the cube. This is considered a nonstructural update. To update captions and descriptions, you can use the **Quick Edits** menu in SAS OLAP Cube Studio, or you can use the UPDATE_DISPLAY_NAMES option for PROC OLAP. This option enables you to identify captions and descriptions on a cube that will change. Here is an example:

```
proc olap cube=cubename description="new description" UPDATE_DISPLAY_NAMES
     DT_TABLE="table name";
      metasvr options;
      dimension time description="New TimeDescription" caption="New TimeCaptions";
```

Note: The only description that can change when a cube is renamed is the cube description (the drill-through table name might also be changed during a rename). Other descriptions and captions must be altered in a separate step after the cube is renamed.

Adding New Members to an Incrementally Updated Cube

By default, the Incremental Update function expects new members and new data to be added to the cube. However, when adding new data to a star schema, you can specify that only data will be added with the NONUPDATEABLE option. When this option is selected, the Incremental Update process does not check the dimension tables for new members. The NONUPDATEABLE option can be specified in the PROC statement or on an individual DIMENSION statement. If the NONUPDATEABLE option is specified in the PROC statement, none of the dimension tables are read. Alternately, you can specify NONUPDATEABLE in the DIMENSION statement if you are adding new members to a specific dimension.

Note: You should set the IGNORE_MISSING_DIMKEYS=VERBOSE option when using the NONUPDATEABLE option.

IN SAS OLAP Cube Studio, you can check the options **Cube Designer - Dimensions** and **Dimension Designer- General**.

Reorganization of Cube Levels

Overview

When you are administering SAS OLAP Cubes, it might be necessary to perform multiple updates of a cube for various business reasons. After a cube has been updated several times, it might be necessary to reorganize the levels for the cube. The deciding factor for this would most likely be a failure of the cube to update successfully, accompanied with error messages that are similar to the following messages:

```
ERROR: The new member name "64  " belonging to the "type" level cannot be
               added to cube because the limit for inserting new captions between
               existing members "  63" and "1001" has been reached.  Consider using
               the proc olap reorganization options to create additional space.
ERROR: A problem was encountered when attempting to update the cube's
               hierarchies with new members.
ERROR: Cube "cubename " cannot be updated.
```

If you cannot update your cube and receive this message, you must reorganize your cube level before you can resume updating your cube.

Why Reorganize?

Overview

Reorganization of a cube must be done after a cube update fails when accompanied by an error message that suggests using "proc olap reorganization options" to create additional space for that level. A level can run out of space for new level members after multiple cube update events. This occurs when new level members have been added to the same sorted location for each event. If too many new level members are inserted into the same sorted location, then eventually cube update fails. Unfortunately, this limit can be reached rather quickly if multiple cube update events take place and, for each event, new level members are inserted into the same location between existing members due to their relative sort order. Reorganization changes the internal structure of the cube so that cube update can once again proceed successfully.

Scenario Example

Consider that you have a sales data cube that contains a level for the customer name, and you have numerous data records for that level where the customer name starts with the letter S. Each time a cube update takes place, a new customer name is added whose name starts with the letter S. There is a finite amount of reserved space for new last names when a cube is first made. If all new names added sort in the same location of the alphabet, rather than being evenly distributed throughout the alphabet, then the odds of running out of reserved space increase with each new cube update event.

Eventually, there is no longer any reserved space left to fit in yet another last name that starts with S. However, there might be ample room to add a name that starts with the letter Q, X, or I. Nonetheless, the customer name level will still have to be reorganized before new names starting with the letter S can be added to the cube.

OLAP Procedure

The following PROC OLAP items are available to reorganize SAS OLAP cubes:

- REORGANIZE_LEVELS | REORG_LEVELS option for PROC OLAP
- REORGANIZE_LEVEL | REORG_LEVEL .

In SAS 9.4, you can reorganize using the **Quick Edits** menu in SAS OLAP Cube Studio.

Multiple Language Support Cubes

You cannot reorganize multiple language support (MLS) cubes. You can, however, update an MLS cube.

Updating Member Properties

When you add data to a cube, the values for member properties are normally not updated. However, you can specify the DIMENSION statement option UPDATE_DIMENSION and the MEMBERS_AND_PROPERTIES value to update the member properties for a dimension. When updating member properties, the new values must be contained in either the ADD_DATA table for a detail load cube or the update table for the dimension for a star schema load.

For a star schema cube, it is possible to change the member property values without adding data to the cube. The table of new data is not required. You can select to update member properties on all the dimensions or on just certain dimensions. In both instances, the original dimension table for a dimension is scanned for new values unless a different table is provided. The Cube Designer – Dimensions page and the Dimension Designer - General page have the option **Allow new members during incremental update**. See the SAS OLAP Cube Studio Help for additional details.

Specifying Drill-Through Tables

Because new data and possibly new members are being added to the cube, a new drill-through table might be required. You can specify a drill-through table that will be associated with the cube after it is updated with the DRILLTHROUGH_TABLE PROC OLAP option or you can specify a new drill-through table in SAS OLAP Cube Studio. Follow these steps to enter a new drill-through table in SAS OLAP Cube Studio.

1. Select **Incremental Update**.
2. Select either **Update In-Place** or **Generate New Cube**.
3. On the Update General page, enter the new drill-through table in the **New drill-through table** field.

Improving Drill-Through Performance

Beginnning in SAS 9.4, it is now possible to reduce the number of columns that are returned for drill-through tables when users perform a drill-through request for an OLAP cube. Prior to SAS 9.4, every column in the drill-through table was returned in the drill-through query results. The OLAP server now performs a more specific query that streamlines the drill-through request. Instead of retrieving all columns from the drill-through table, only the columns that correspond to cube levels are returned, and only those columns relevant to the measures being requested in the drill-through query are returned.

Furthermore, the columns of drill-through data that are returned appear in a predictable order that corresponds to the cube definition. As has always been the case, all returned columns reflect any permission conditions that have been set on the drill-through table.

This change improves the performance for those drill-through tables that contain columns that are not relevant to the cube. In addition, the query results are displayed in the dimension order, which simplifies the interpretation of the query results.

For example, here is a cube definition with three dimensions and two measures. The cube dimensions are defined in this order:

```
 Dimension Time:   levels = Dte
 Dimension Car:    levels = Car, Color
 Dimension Dealer: levels = Dealer, Dest
 Measures :       = SalesSum ,ReturnsSum ( for the vars SALES and
RETURNS
respectively)
```

For this hypothetical cube, the drill-through table for the cube contains nine variables that are listed in this order: DTE, CAR, COLOR, DEALER, TYPE, SALES, RETURNS, WGT, and DEST. Here is an example of a possible MDX drill-through query for this cube:

```
drillthrough
   select
    [DATE].[All DATE].[March] on 0,
    [CARS].[All CARS].[Ford] on 1
   from mddbcars
where measures.sales_sum
```

Here are the results of the drill-through query in SAS 9.4. The columns for DATE and CARS are returned, as well as all other cube levels, and only the SALES measure is returned. All levels in the cube are returned in an order that corresponds to the dimension order of the cube definition. Any columns that are not relevant to the cube definition have not been returned. In this example, the columns for TYPE and WEIGHT are not returned.

Notice also that only the members " March" and "Ford" are returned for DATE and CARS, since those are the specific members requested in the MDX drill-through query.

```
dte    car    color   dealer     dest    sales
----   ----    -----  ------     ----    ------
March Ford     White    Smith    AL       15000
March Ford     Blue     Smith    NJ       10000
March Ford     Green    Finch    NC       17000
```

Prior to SAS 9.4, the same drill-through query would have returned the following output. Notice that the order of the variables returned is random with respect to the cube definition, and some columns of data are returned that are completely extraneous to the cube definition:

```
date   car   color  dealer type sales returns  wght                dest
---    ---   -----  -----  ---- ------ -----   -----------------   ----
March  Ford  White  Smith   1   15000  45      0.900000000000003    NC
March  Ford  Blue   Smith   1   10000  110     0.900000000000003    AL
March  Ford  Green  Finch   2   17000  400     0.900000000000003    NJ
```

Drill-Through Tables and Aggregated Columns

When you have a cube whose NWAY is defined by an AGGREGATION statement, the MEASURES statement will contain an AGGR_COLUMN option that indicates the column in the table that contains the aggregated measure data for that MEASURE.

Here is an example:

```
aggregation DTE CAR COLOR DEALER DEST / table=olapsio.carnway;
measure SALES_SUM  stat=sum aggr_column=SSALES format=dollar15.2;
```

Starting with SAS 9.4M2, if you are using drill-through with the cube described above it is advisable to additionally include the COLUMN= option on the MEASURES statement when indicating an aggregated column. This enables you to indicate the column in the drill-through table to use for drill-through execution. Here is an example:

```
measure SALES_SUM  stat=sum column=SALES aggr_column=SSALES
format=dollar15.2;
```

It is possible that, if you have not specified the COLUMN= option, your query will return an exception that identifies the measure or the level that cannot be found.

NWAY Considerations

When you use the Incremental Update function on a cube, the cube should have a MOLAP NWAY. If the cube is loaded from a fully summarized table, the Incremental Update function is not allowed. The NO_NWAY option is also of importance. When using PROC OLAP to add data the cube, you must have the NO_NWAY option set

to NO. If the NO_NWAY option is set to YES, it is assumed that the cube does not have a MOLAP NWAY and adding data is not allowed.

Updating Multilingual Cubes

For multilingual star schema cubes, the original dimension tables are always used. When you update a single-language star schema, if a table is not specified for the dimension table, the original dimension table is scanned for new members.

If you are updating a multilingual cube, you cannot modify the languages that are selected for the cube. To add new members, the original dimension tables (including the different language tables) must be updated with the new members. For multilingual cubes, the original tables are always re-read.

To update multilingual captions, specify the OLAP procedure options MLSCAPUPD and UPDATE_DISPLAY_NAMES, as described in in “MLSCAPUPD” on page 325..

In SAS OLAP Cube Studio, select **Quick Edits** ⇨ **Update Captions**.

See also “Multiple Language Support for Cubes” on page 104 .

Format Search Path and SAS Source Code Considerations

When updating cubes that have added syntax for a user written formatted search path or SAS code, you must consider how that information will be updated as well. Here are some possible scenarios.

Table 8.1 *Formatted Search Path Scenarios*

Update Process	Update Results
You perform an update in-place and do not make any changes to the format search path or SAS source code.	The metadata for the cube does not need to be updated. When the update runs, the current format search path and SAS source code must be submitted with the code.
You perform an update in-place and change the format search path or the SAS source code.	The new values for the format search path and SAS source code will be updated in the metadata for the cube in addition to being submitted with the code or saved with the exported code.
You create a new cube in the update and do not make any changes to the	The new generation of the cube will be created using the same format search path and SAS source code as the original cube, and its metadata should also include the

format search path or the SAS source code.	same format search path and SAS source as the original cube. The submitted code or exported code will contain the current format search path and SAS source code. PROC OLAP will copy the current format search path and SAS source code to the new cube as part of the update.
You create a new cube in the update and enter a new format search path or new SAS source code.	The next generation of the cube will be created with the new format search path and SAS source code, and its metadata will include the new format search path and SAS source code. The submitted or exported code will contain the new format search path and new SAS source code. However, PROC OLAP will copy what is currently in the metadata for the original cube.

Exporting Cubes That Have Been Updated

After you update a cube, it might be necessary to export that cube for use in another SAS OLAP environment. If a cube exists, and if it has been updated, and if it has been coalesced, then it can be exported.

Note that you can export generations of the cube other than the current (latest) generation..

Input Data Tables for Cube Updates

When you add data to a cube, you must select a table that contains the new data for the cube. This might be new data for existing members or new members with new data. The table must be defined in the metadata repository and it must contain all the columns for the measures. Depending whether you are incrementally updating the cube or rebuilding the cube, you must manage the input data accordingly. If you are updating the cube incrementally, then your table does not need to contain all the cumulative data. However, if you are rebuilding the cube, then your input tables must contain all the data from the original build and all the incremental updates.

If you are rebuilding a cube, it is very important that the data used as input to the rebuilt cube contain all the incremental updates that have been made since the original cube was created. This can be done one of two ways:

1 The input base or fact table definition in the SAS metadata should be a view that combines the original input table plus all subsequent update tables into a single logical table. This is the recommended approach.

2 The input base or fact table itself is physically updated to add data from all update tables.

It is not possible for PROC OLAP or SAS OLAP Cube Studio to verify the correctness of the input data. This is the responsibility of the administrator or cube builder. Depending on what data source the cube is loaded from, some requirements apply:

Table 8.2 *Input Table Considerations*

Data Source	Requirement
Detail table	All the level columns must be in the new data table.
Star schema	The foreign key columns must be in the new data table. However, the level columns must be in the dimension table along with the unique key columns. For star schema cubes, you do not have to select a source table. If a source table is not selected, then only the dimension tables will be processed and only if the UPDATE_DIMENSION option is set to MEMBERS or MEMBERS_AND_PROPERTIES for the dimension. No new data is added to the cube. In this case, the ADD_DATA option should not be added to the PROC OLAP statement. You can, however, still specify new dimension tables and specify a value for the UPDATE_DIMENSION option. This is valid only for dimensions in a star schema that originate from a dimension table. Dimensions in a star schema that come from the fact table can be updated only if a source table is selected and ADD_DATA is specified in the PROC OLAP statement.
Fully summarized table	The cube cannot be updated.

When you add data to a cube, the cube and its corresponding tables must be kept in sync. When you add new members to the cube using the Incremental Update process, the original tables are re-read by PROC OLAP unless a different table is provided.

Note: If the original dimension table for a star schema is updated with the new members, you do not need to select a table for a dimension.

Schema and Repository Considerations

When you add data to a cube, a new name or a new OLAP schema must be provided for the updated cube. By default, the OLAP schema of the original cube is selected. In addition, the new generation of the cube should be created in the same repository as the original cube. If you are defining the repository with the PROC OLAP METASVR statement, use the repository for the cube that is being updated. Do not use the default repository.

Physical Storage and Metadata Considerations

The physical files that compose SAS OLAP cubes are contained within two subfolders:

- the cube path folder
- the generation path folder

The cube path folder name matches the cube name, under the folder defined by the PATH= option. The aggregation racks (tables) are located in the cube folder. The remaining internal cube files are placed in a generation folder. Each generation will have a new generation folder, with subsequent names (gen0001, gen0002, gen0003, and so on). These folders will be created under the existing cube folder.

Connecting to a Workspace Server

Because adding data to a SAS OLAP cube involves submitting SAS code to accomplish the actual update, a connection to the workspace server must be established. If a connection cannot be made, you can continue using the Incremental Update function and save the generated PROC OLAP code to submit later in a SAS session.

PROC OLAP Options

Depending on the type of cube update you are performing (in-place or incremental), the following PROC OLAP statements and options can be used:

- ADD_DATA
- NONUPDATEABLE
- OUTCUBE
- OUTSCHEMA
- RENAME
- UPDATE_DIMENSION
- UPDATE_DISPLAY_NAMES
- UPDATE_INPLACE

PROC OLAPOPERATE Options and SAS OLAP Server Monitor

In addition to PROC OLAP options, it might be necessary to use PROC OLAPOPERATE options when promoting an incremental update of a cube online. You can use the PROC OLAPOPERATE options DISABLE CUBE and then ENABLE CUBE, or you can perform these functions with the SAS OLAP Server Monitor plug-in for SAS Management Console.

- DISABLE CUBE
- ENABLE CUBE .

Updating Cubes in SAS OLAP Cube Studio

The Incremental Update function in SAS OLAP Cube Studio enables you to add new members and new data to a cube as well as update captions for dimensions, hierarchies, levels, member properties, and measures. The Incremental Update function enables you to specify cube updates and submit the generated procedure code without rebuilding the cube. Although you can add data to a cube, the cube

structure cannot be modified. Adding or removing dimensions, levels, measures, hierarchies and properties is not part of the Incremental Update function.

When you initially create a cube, you can establish whether the cube will allow new members to be added to the cube at a later point. You can also allow new members to be added to a specific dimension on a cube. The Cube Designer – Dimensions page and the Dimension Designer – General page on the Cube Designer wizard have the option **Allow new members during incremental update**. See the SAS OLAP Cube Studio Help for additional details.

The Incremental Update wizard is available from the **Actions** menu and the context menu in SAS OLAP Cube Studio. It is active on the menu if the selected cube meets the requirements for adding data. If you select the Incremental Updates function, a drop-down menu opens. Depending on the type of update that you need to perform, select one of the following options.

- **Update In-Place**
- **Generate New Cube**
- **Coalesce Incremental Data**

After you have updated your cube, you can check to see a list of the generations for the cube. Select the **Generations** tab on the Properties dialog box for the cube.

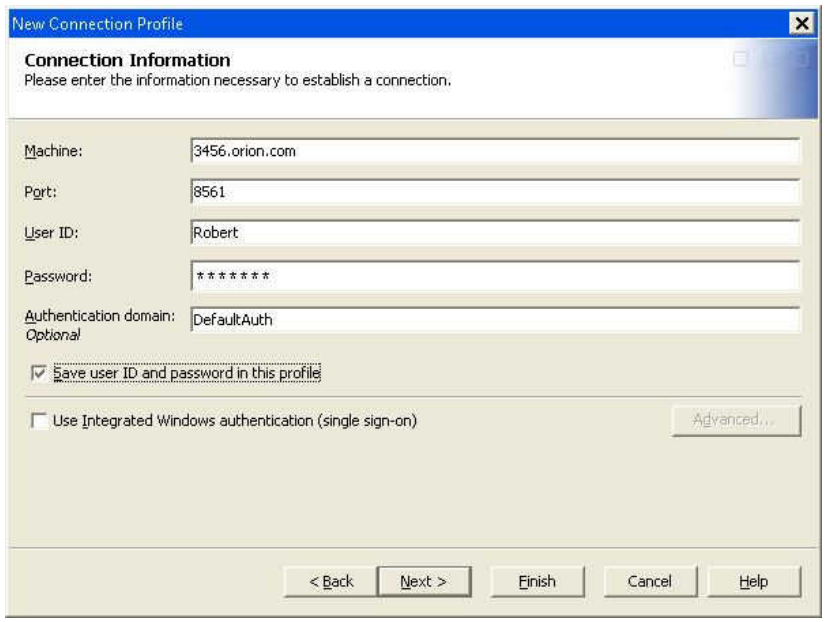
See the SAS OLAP Cube Studio Help for additional details about the Incremental Update function.

9

Cube Building and Modifying Examples

Defining a Connection Profile

Each time you log on to SAS OLAP Cube Studio, a connection is made to a metadata server. Before you can build a cube in SAS OLAP Cube Studio, you must specify a connection profile that contains the information for the metadata server that you want to build your cube with. In SAS OLAP Cube Studio, select **File** ⇨ **Connection Profile**. On the Open a Connection Profile dialog box, you can choose to either create a new connection profile or edit an existing one. If you choose to create a new connection profile, the Connection Profile wizard will open. The following display shows the information that is entered for a connection profile.



Enter the machine information for the metadata server that you will connect to and retrieve a data source from.

- In the New Connection Profile dialog box, enter the machine ID, port, your user ID, and password.

  Note: These fields are the equivalent of the following METASVR statement options:

- HOST=
- PORT=
- USERID=
- PW=

  - Here is an example of the PROC OLAP code:

```
metasvr host=localhost
```

```
port=9999
protocol=bridge
userid=userid
pw=pw
repository=Foundation
olap_schema="OLAP Schema"
```

After you have entered the basic connection information, you can select to do one of the following:

- enter the authentication domain for the SAS Metadata Server
- save the user ID and password with the connection profile use Integrated Windows authentication (single sign-on). The Integrated Windows authentication option specifies that access is based on the user's previous authentication to the Windows desktop.

Note: For more information about authentication and single sign-on, see "Understanding Authentication" in the *SAS Intelligence Platform: Security Administration Guide*.

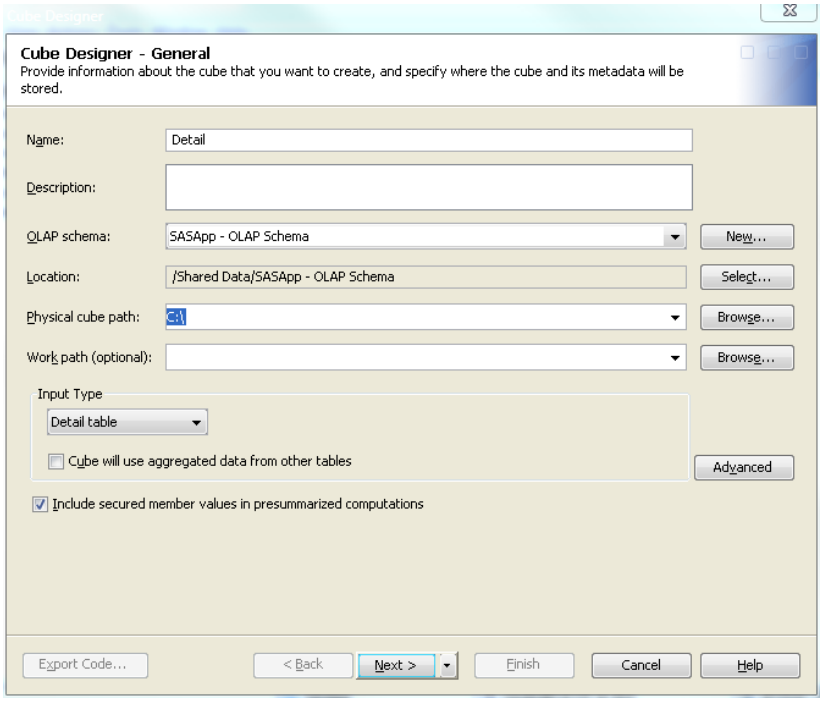
Building a Cube from a Detail Table

Overview

A detail, or base, table is a table whose data pertains to a single area of interest. It is any table defined in a SAS Metadata Repository that contains the measures and levels for a cube. You can build an OLAP cube from a detail table by using the Cube Designer wizard in SAS OLAP Cube Studio. In this example, you use data from a product marketing campaign. You establish measures and summaries of product statistics, geographic location of potential customers, and revenue.

Enter General Cube Information

After you have established a connection profile, you can begin to create a cube. Select **File** ⇨ **New** ⇨ **Cube**. On the Cube Designer – General page, enter the basic cube information. For this example input type, you select **Detail Table**. The following display shows fields that you enter information for.



- Enter information in the following fields:
  - **Name**
  - **Description**
  - **OLAP schema**
  - **Location** (SAS folder)
  - **Physical cube path** (path in the file system to store the cube)
  - **Work path** (path for temporary work files)
  - **Input Type** (detail table)

Select a Detail Table

Overview

On the Cube Designer – Input page, select a base or detail table for your cube. If one does not exist for your data, click **Register Table**, and then define the source from which you will import your metadata.

Drill-Through Table

On the Cube Designer – Input page, you can also select or define an optional drill-through table. Drill-through tables can be used by client applications to provide a view from processed data into the underlying data source.

If a drill-through table does not exist for your data, click **Register Table**, and then define the source from which you will import your metadata. The following display shows the base table and drill-through table that are selected for this cube.

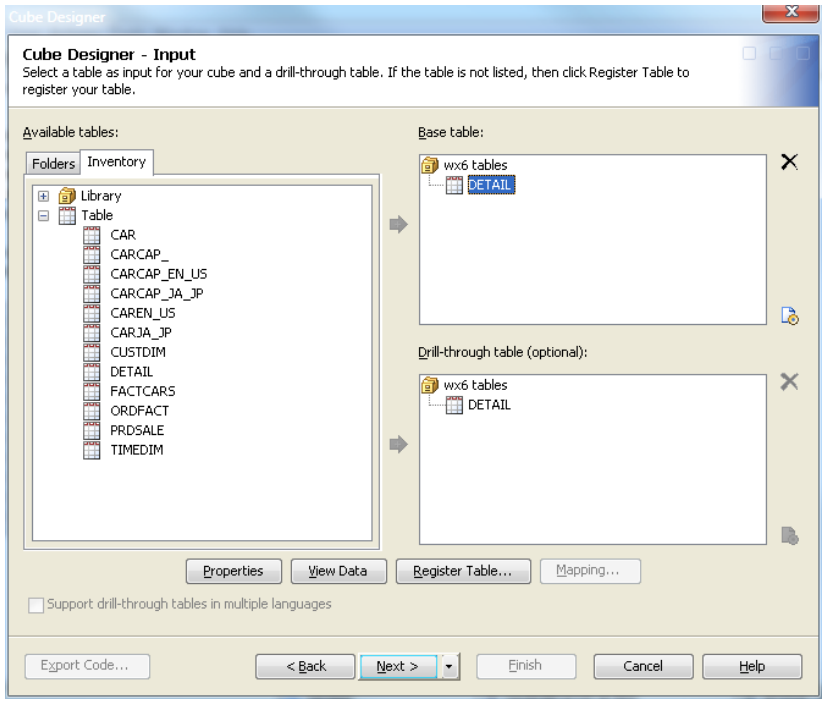


Table Options

On the Cube Designer – Input page, you can specify the data set options that are used to open either the detail table or drill-through table for your cube. Click the  button next to either the **Base table** or **Drill-through** trees. The **Table Options** button opens the Table Options dialog box, which enables you to specify data set options that are used to open the data set. For example, you could enter a WHERE clause or other information to subset the selected table. The options are stored as part of the cube and then reapplied when the data is accessed at run time. You can also specify data set options in the Dimension Designer – General dialog box (for use with star schemas) and the Stored Aggregates dialog box (for use with summarized tables). For more information, see “Data Set Options” in the *SAS Language Reference: Concepts*.

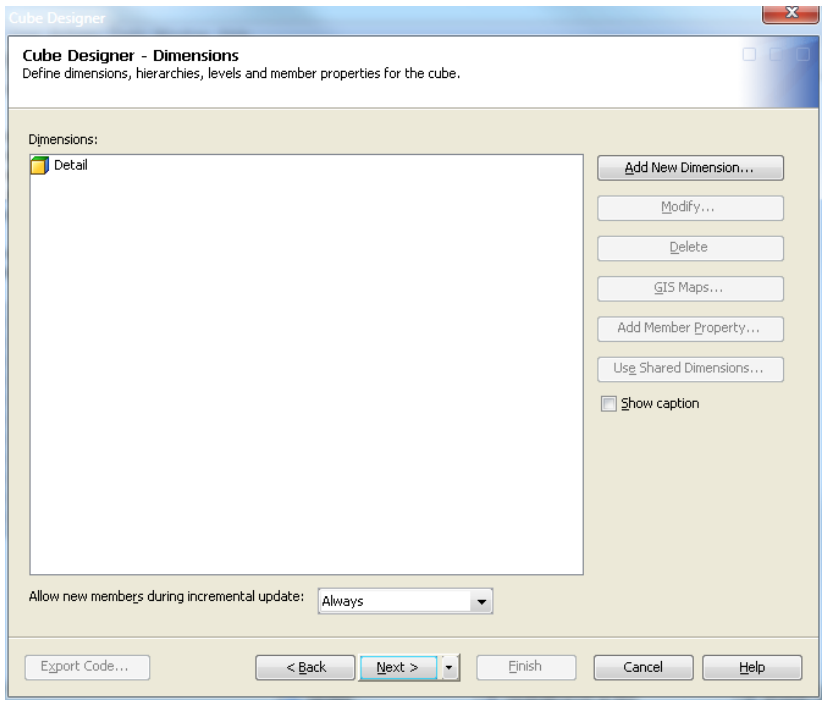
Define Dimensions, Levels, and Hierarchies

Overview

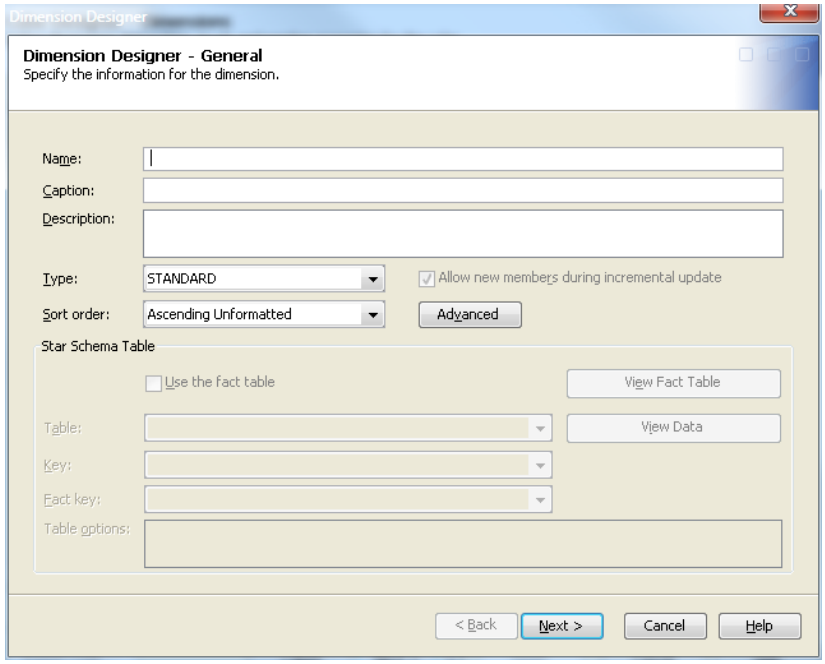
Now that your basic metadata server and cube information has been entered, you can define the different dimensions and their respective levels and hierarchies. For this example, the following dimensions are created:

- Products
- Dates
- Geography
- Customers
- Orders

At the Cube Designer – Dimensions page, click **Add New Dimension**.



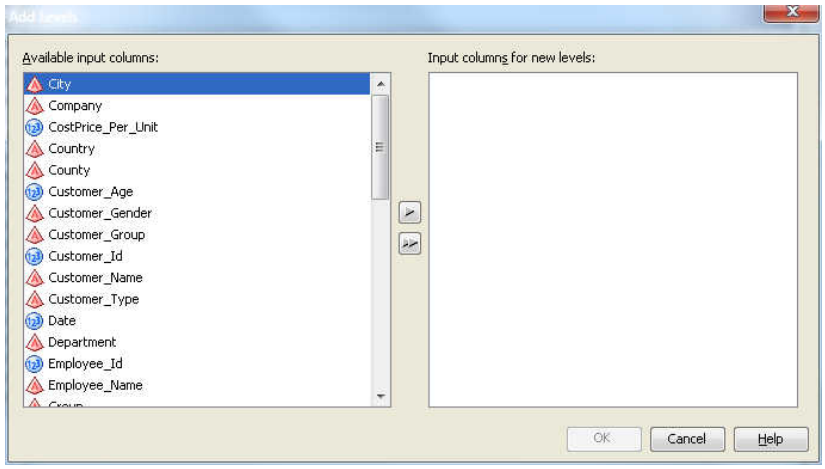
This opens the Dimension Designer – General page, as seen in the following display.



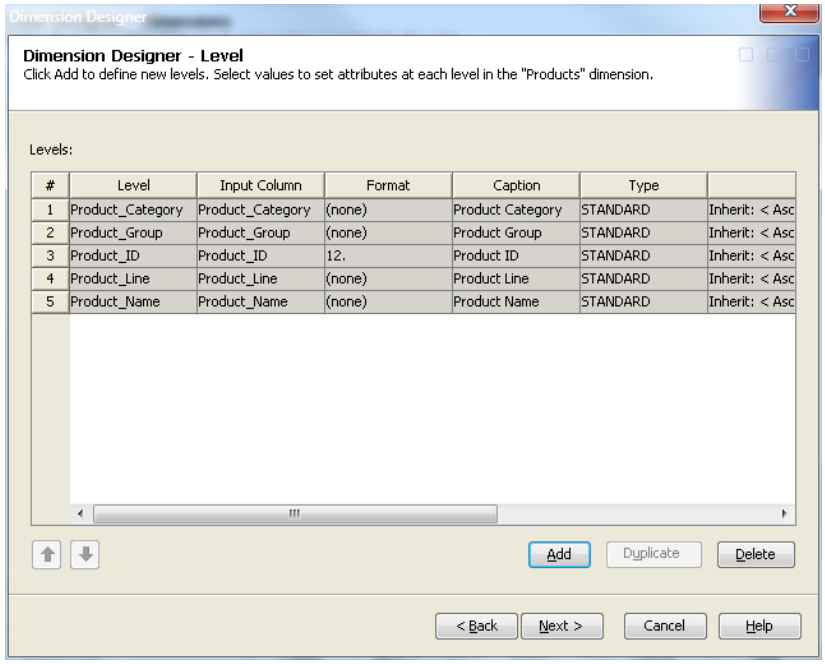
Enter the information in the following fields:

- **Name**
- **Caption**
- **Description**
- **Type** (STANDARD, GEO, or TIME)
- **Sort Order**

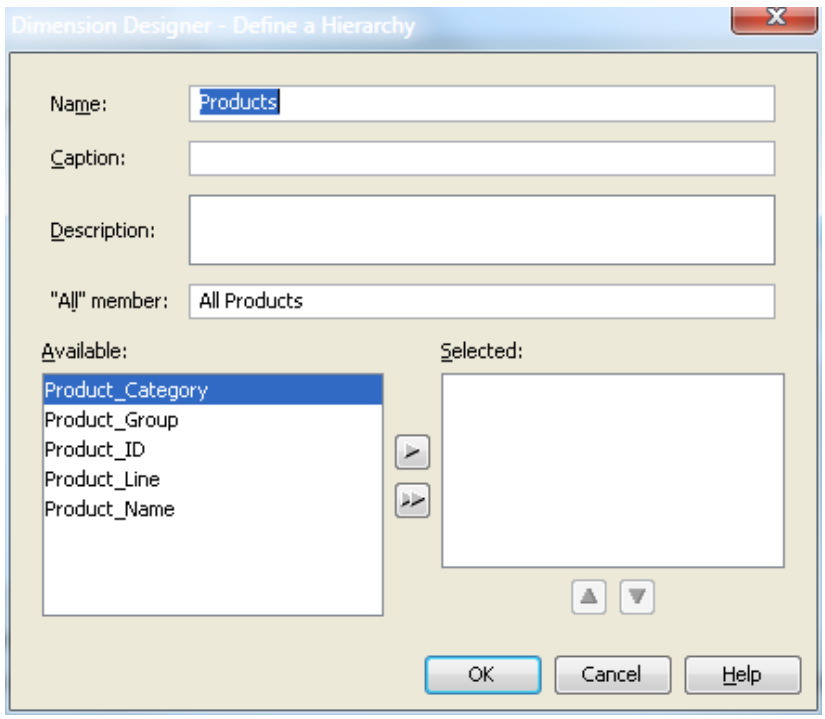
Click **Next** to open the Dimension Designer – Level page. Select **Add** to open the Add Levels page, as seen in the following display.



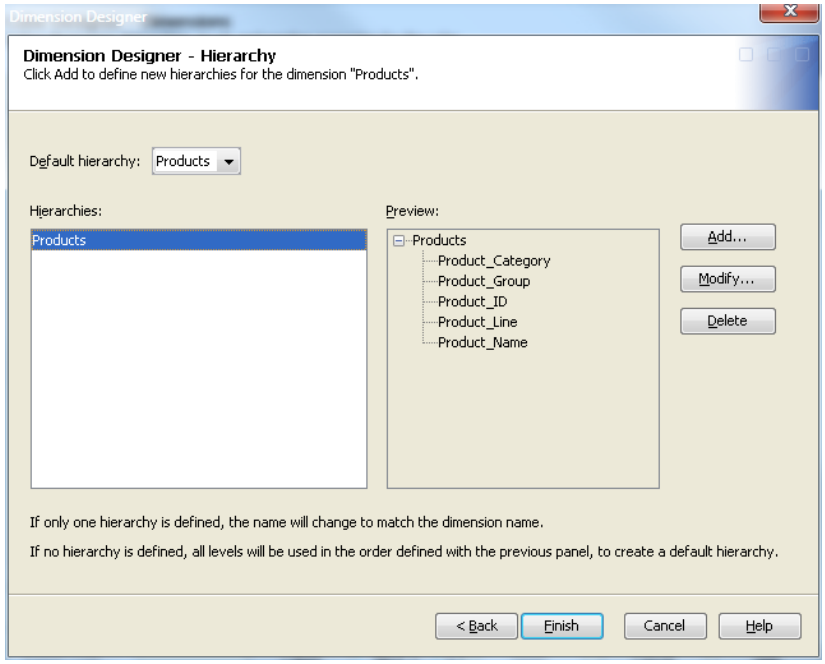
Select the levels that you want to add to the dimension. Select **OK** to return to the Dimension Designer – Level page, where the selected levels are listed. You can now define properties such as format, time type, and sort order for the levels that you have selected. See the following display.



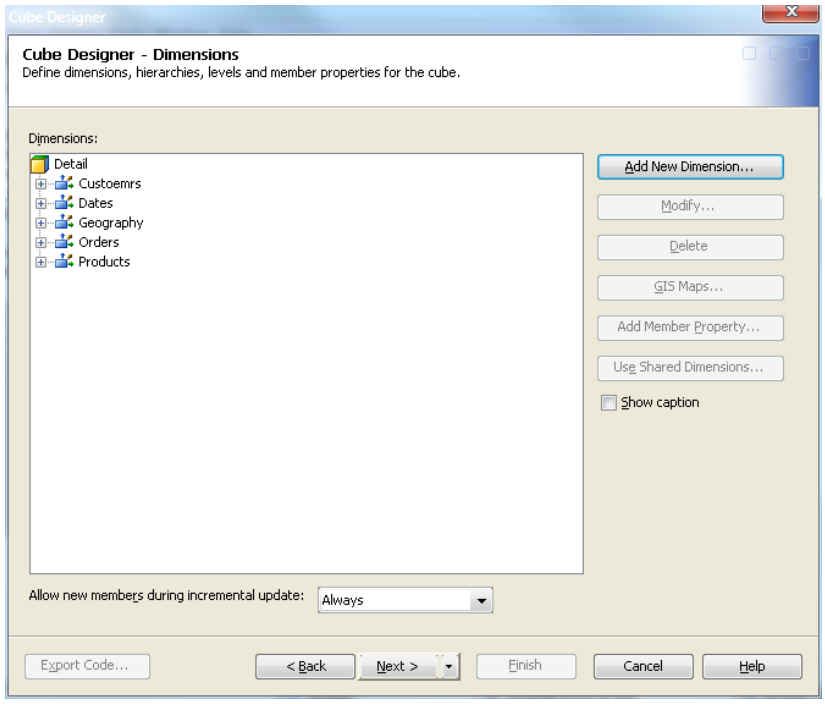
Next, define hierarchies for the levels on the Dimension Designer – Hierarchy page. You can click **Add** to open the Define a Hierarchy page and individually select the levels for the hierarchy.



Or you can click **Finish** on the Dimension Designer - Hierarchy page to accept the order of the levels that are defined on the previous Dimension Designer – Level page. If you select this option, the hierarchy is assigned the same name as the dimension. See the following display.

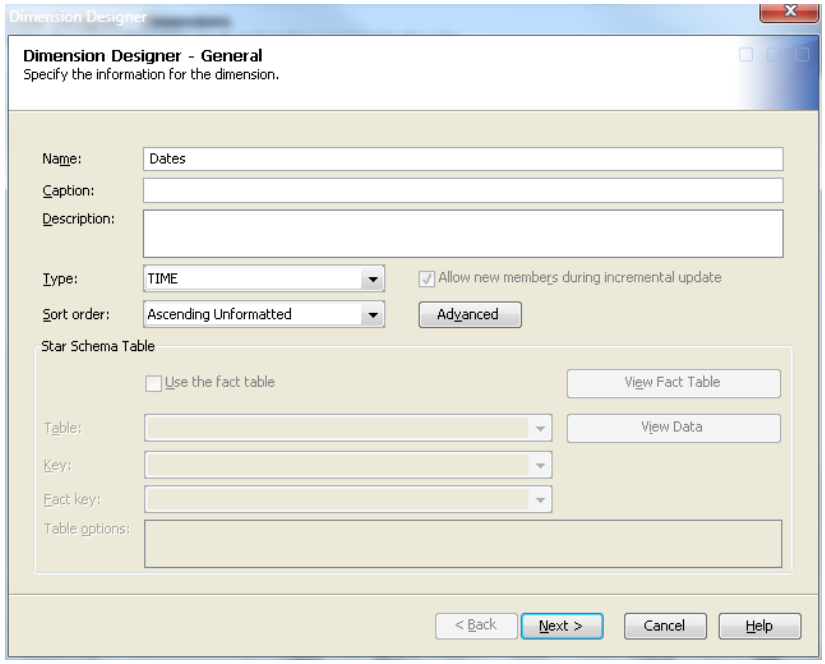


Repeat this process for each dimension. After you create each dimension, it is listed in the **Dimensions** panel of the Cube Designer – Dimensions page. See the following display.



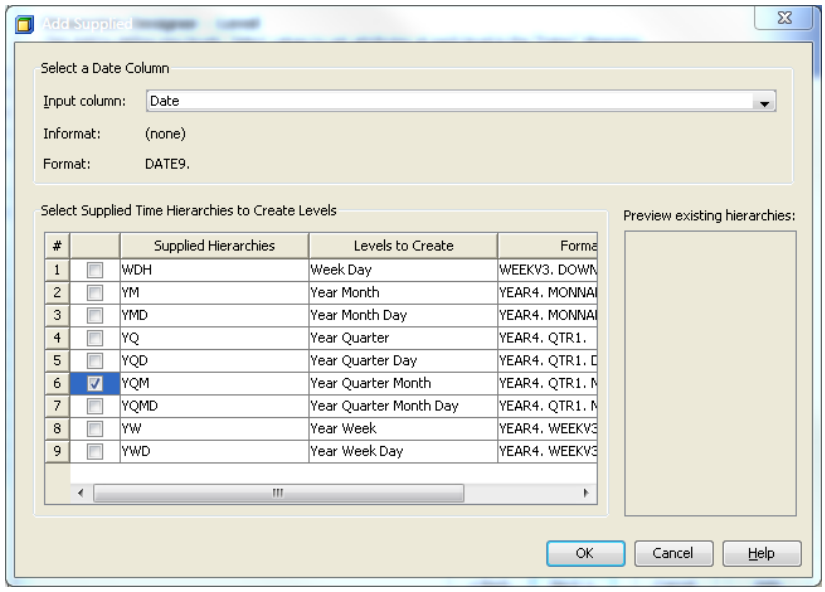
Creating a Time Dimension

When you create the Dates dimension, you must specify the **TIME** dimension type on the Dimension Designer – General page. See the following display.

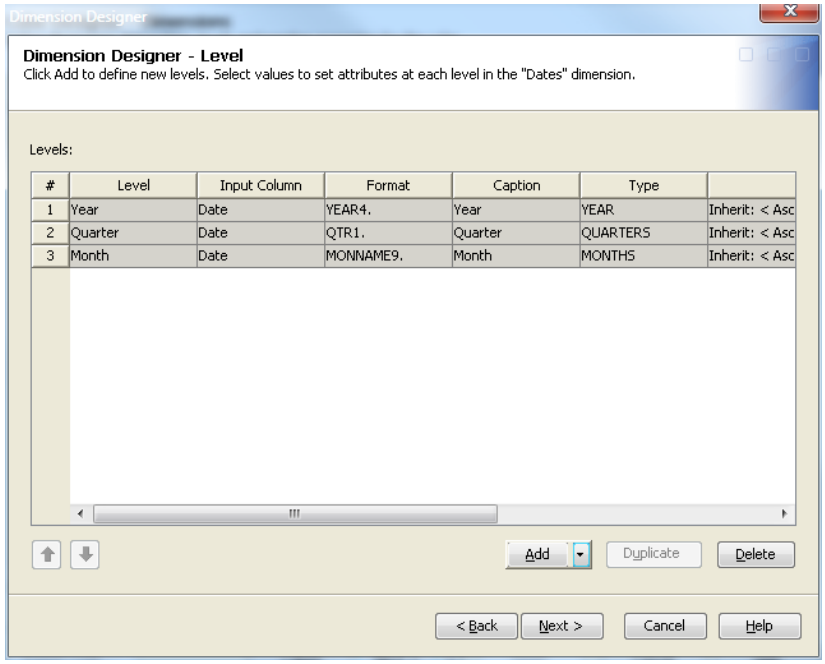


Specifying the TIME dimension type enables **Add supplied time hierarchies** on the Dimension Designer – Level page. The **Add** button is converted to a drop-down list of options. The **Add levels** and **Add supplied time hierarchies** options are now available for selection.

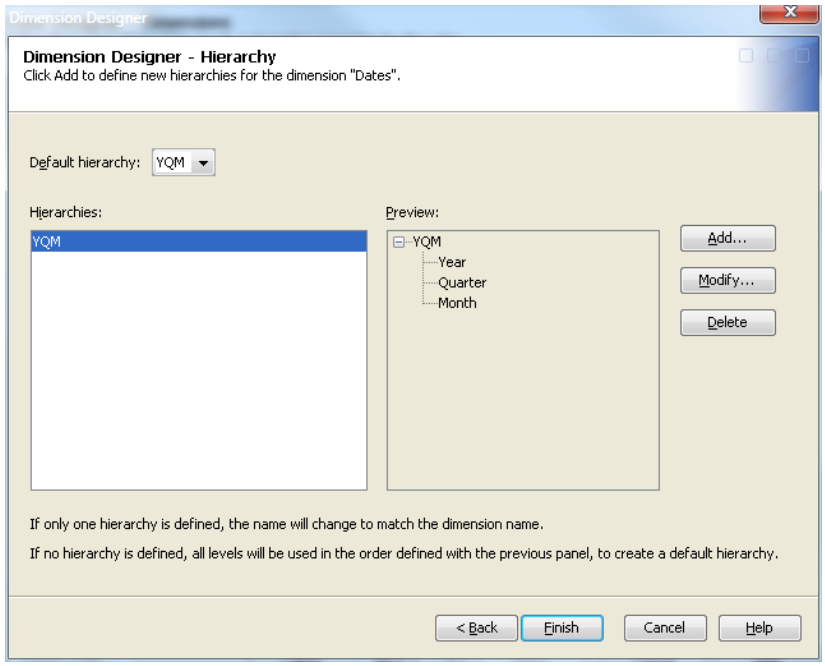
Select **Add supplied time hierarchies**. This opens the Add Supplied dialog box. Select from the list of supplied time hierarchies to create the time levels. This also creates the hierarchies for the dimension. See the following display.



You can then define properties such as time type and sort order for the levels that you have selected. See the following display.



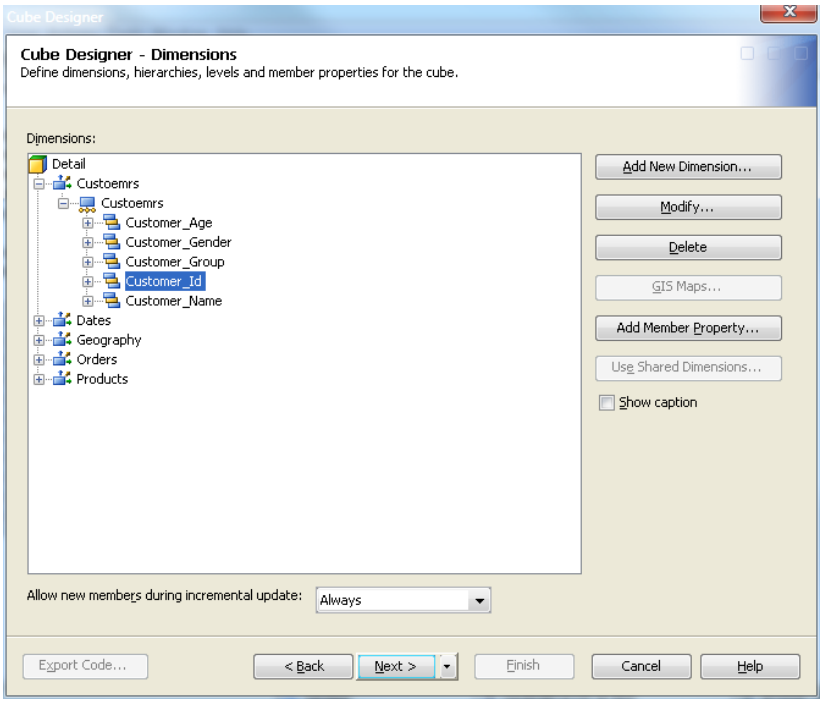
The hierarchy or hierarchies that are selected on the Add Supplied dialog box are listed in the **Hierarchies** panel on the Dimension Designer – Hierarchy page. If there is only one hierarchy, as with this example, the hierarchy name is changed to match the dimension name. See the following display.



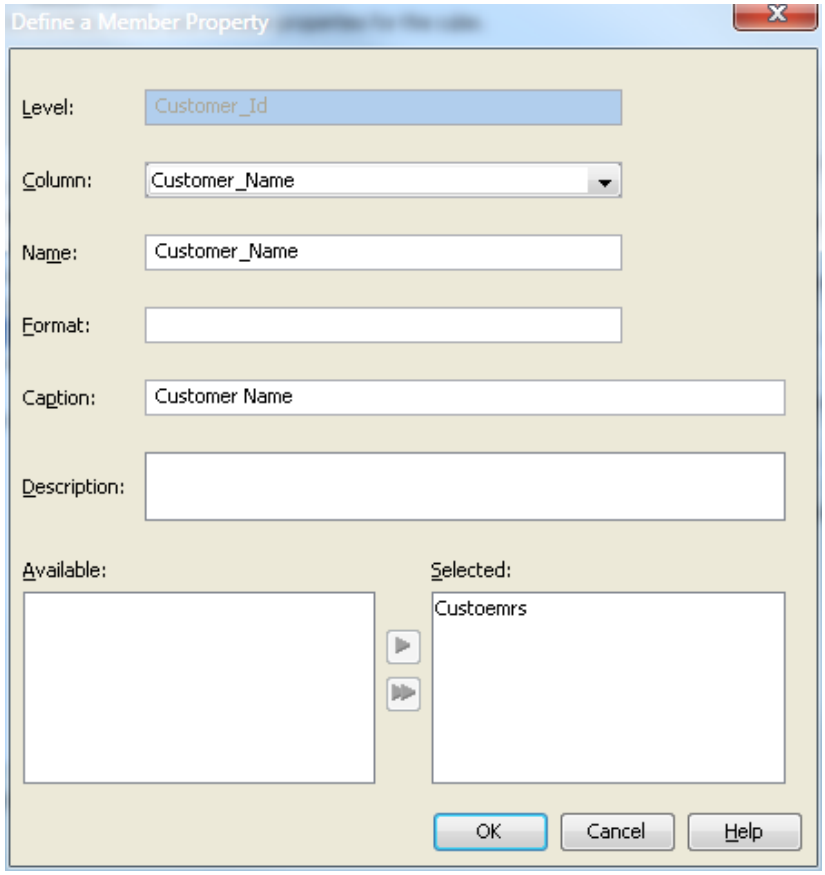
Define Member Properties

You can now define the member properties for any needed cube members. A member property is an attribute of a dimension member. A member property is also

an optional cube feature that is created in a dimension to provide users with additional information about members. For this example, you can define the customer gender as a member property. Define member properties in the Cube Designer – Dimensions page. Select the **Customer ID** level, and then click the **Add Member Property** button, as seen in the following display.



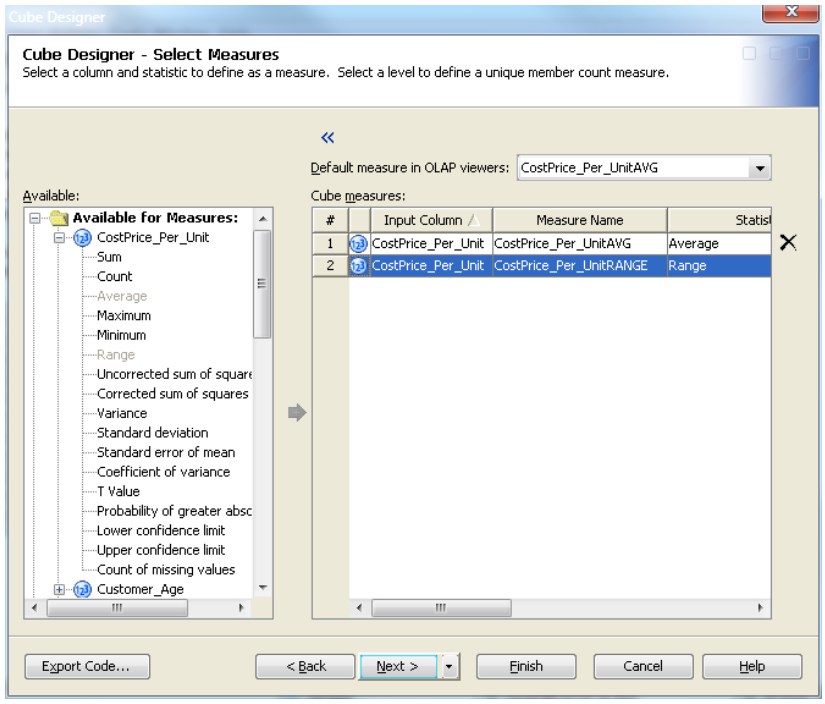
At the Define a Member Property page, enter the member property column name, format, and caption.



Define Measures

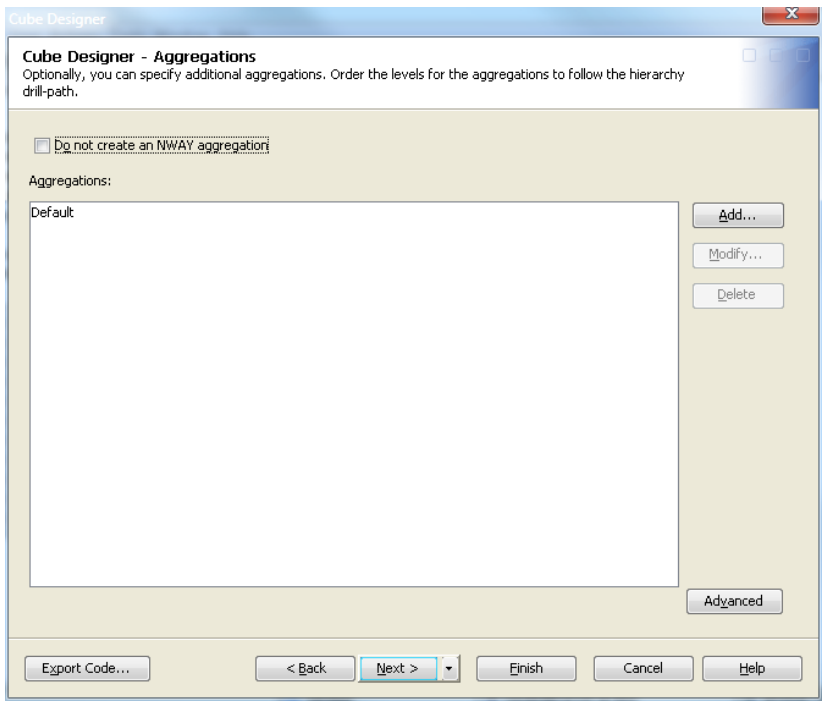
You can now define the measures for the cube. In this example, you define measures for the CostPrice Per Unit. Define the measures for the cube at the Cube Designer – Select Measures page.

Modify any measure attributes such as measure captions and formats as shown in the following display.

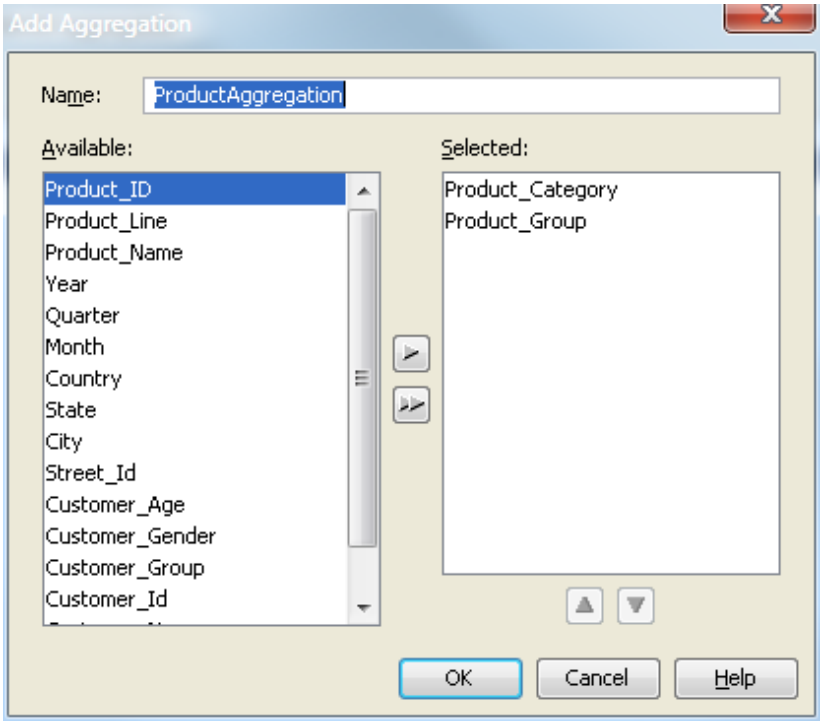


Define Aggregations

You can now define the aggregations for the cube. Aggregations are summaries of detailed data that are stored with a cube or referred to by a cube. They can help contribute to faster query response. Define the aggregations for the cube from the Cube Designer – Aggregations page, as shown in the following display.



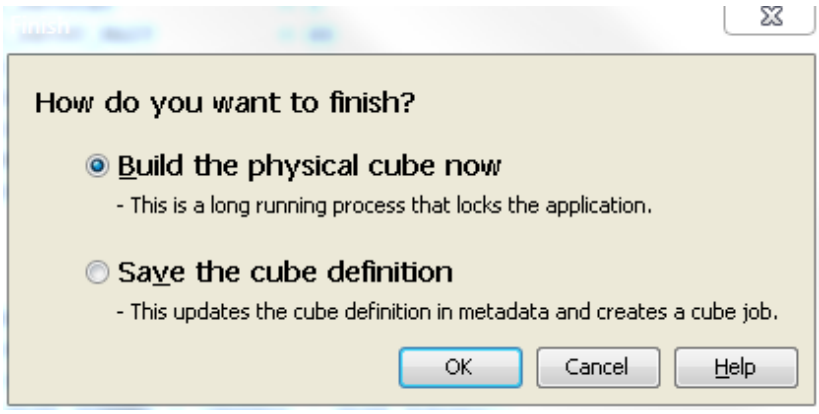
Select **Add** to specify a user-defined aggregation. This opens the Add Aggregation dialog box, as shown in the following display. In this dialog box, you can select levels to add to the aggregation that you are defining.



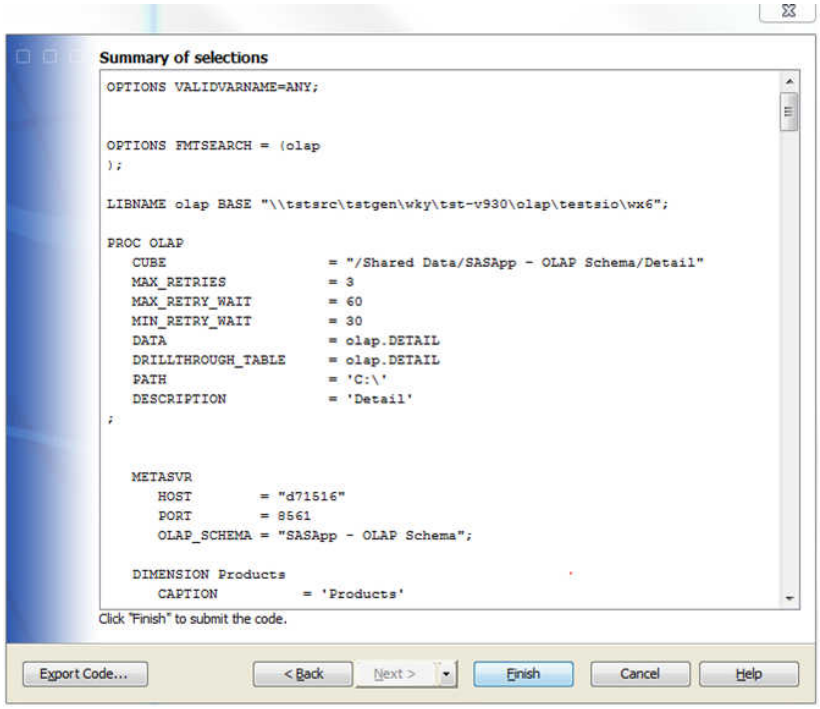
Select **OK** to return to the Cube Designer – Aggregations page, where the new aggregation is listed. Select **Next** to go to the Cube Designer – Finish page.

Build the Cube

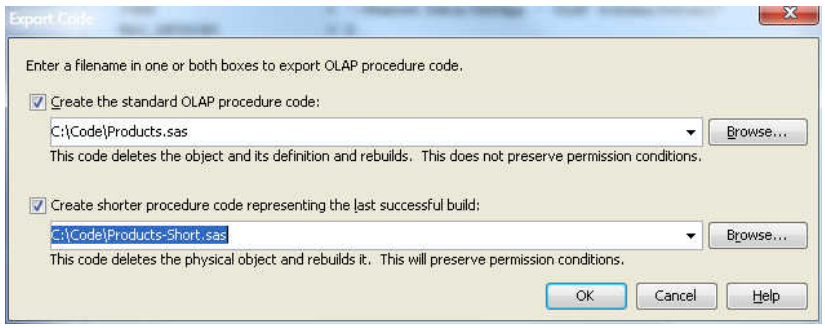
You can now build the cube. On the Cube Designer – Finish page, review the settings for the cube, and then click **Finish**. The Finish dialog box is displayed as shown in the following display.



Choose to build the physical cube or to generate cube metadata without building the physical cube. Click **OK** to save your selection and display the Summary of Selections page, as shown in the following display.



On the Summary of Selections page, you can review the specifications for your cube. At this point, you can click **Back** to make changes, **Finish** to build the cube, or **Export Code**, as shown in the following display.



The Export Code dialog box displays the current settings that determine whether, how, and where you store generated SAS code. To export code for a new cube or to export code for an existing cube without saving permission conditions, click **Create the standard OLAP procedure code**.

To generate shorter OLAP procedure code and to preserve existing permission conditions, click **Create shorter procedure code representing the last successful build**.

To not save the generated code, deselect both of the check boxes.

Enter the file location(s) where you want to save your generated code, and then click **OK** .

SAS OLAP Code Saved in Export

When you export the code that is generated in a cube build, you save the following types of code:

- the SAS LIBNAME statement
- any FMTSEARCH statements
- any additional SAS code
- the PROC OLAP statement
- the METASVR statement
- all other PROC OLAP statements

Code for permission conditions is either preserved or deleted from the export file, as determined in the Export Code dialog box.

You can access the Export Code dialog box by using one of the following methods:

1. On the tree view in SAS OLAP Cube Studio, right-click on a cube and click **Export Code**.
2. In the Cube Designer – Finish page, click **Export Code**.

The Export Code dialog box appears. You can choose to save either the long or short form of the code, or both. Enter the file location or locations where you want to save the resulting code. Click **OK** when finished.

PROC OLAP Statements and Options for a Detail Table

The PROC OLAP code that is generated when a detail cube is built is listed below. A detail cube is unique in that it uses the DATA= option to specify the data source for the cube. The statements each have options that are either required or optional, depending on the cube structure.

Table 9.1 *Statements and Options Used to Load Cubes from a Detail Table*

Statements	Options	Required or Optional
PROC OLAP	DATA=	Required
	CUBE=	Required
	PATH=	Required
	DESC=	Optional

Statements	Options	Required or Optional
	NO_NWAY	Optional
	WORKPATH=	Optional
	DT_TABLE	Optional
METASVR	OLAP_SCHEMA=	Required
	REPOSITORY=	Optional
	HOST=	Optional
	PORT=	Optional
	PROTOCOL=	Optional
	USERID=	Optional
	PW=	Optional
DIMENSION	HIERARCHIES=	Required
	DESC=	Optional
	CAPTION=	Optional
	TYPE=TIME	Required only for TIME dimensions
	SORT_ORDER=	Optional
LEVEL		The LEVEL statement is optional unless you want to specify time periods for each level in a TIME dimension. If you specify a time period for one level, then you must specify a time period for all levels. To specify a time period, you use the TYPE= option.
HIERARCHY	LEVELS=	Required
	DESC=	Optional
	CAPTION=	Optional
MEASURE	STAT=	Required
	COLUMN | ANALYSIS=	Required

Statements	Options	Required or Optional
	AGGR_COLUMN=	Required if you use the AGGREGATION statement with the TABLE= option
	DESC=	Optional
	CAPTION=	Optional
	UNITS=	Optional
	FORMAT=	Optional
	DEFAULT=	Optional
PROPERTY	prop-name	Required
	LEVEL=	Required
AGGREGATION		The AGGREGATION statement is optional unless you are creating additional aggregations. In that case, you must specify the names of the contiguous levels to be used to create the aggregation. Use the TABLE= option for cubes that contain aggregated data from tables other than the input data source.

PROC OLAP Example for a Detail Table

In addition to building a cube in SAS OLAP Cube Studio, you can build an OLAP cube with PROC OLAP code and execute the code in a SAS session. Running PROC OLAP registers your cube and its sources in a metadata repository. It also creates the files that make up the cube. These are the possible input types for an OLAP cube that is built from a detail table:

- a data table (specified in the PROC OLAP statement DATA= option)
- dimension tables (specified in the DIMENSION statement DIMTBL= option)
- presummarized tables (specified in the AGGREGATION statement TABLE= option)

In this example, you use data from a product marketing campaign. For this cube, you establish measures and summaries of product statistics, geographic location of potential customers, and revenue.

1 **Define the metadata profile and general information.** You use the PROC OLAP and METASVR statements here. The detail table is specified in the PROC OLAP statement DATA= option. The CUBE= option is used to specify the metadata folder location of a cube and the name of the cube. The folder must already exist in the metadata. The PATH= option specifies the physical or logical path to the location of a new cube. Within the specified path, the cube is stored in a directory that uses the name of the cube in uppercase letters.

The METASRV statement is used to establish the metadata connection. It identifies the metadata repository in which existing cube metadata information exists or in which metadata about a new cube should be stored. The statement is also used to provide a user's identification and password for the identified repository. Also, the DRILLTHROUGH_TABLE= option is used here to indicate the drill-through table. Drill-through tables are optional and can be used by client applications to provide a view from processed data into the underlying data source.

```
proc olap data=olapsio.detail
   cube=/cubefolders/mycubefolder/Campaign1'
   path="c:\cubes"
   ;
metasvr host=localhost
   port=9999
   protocol=bridge
   userid=userid
   pw=pw
   repository=Foundation
   olap_schema="OLAP Schema"
   ;
```

2 **Define dimensions, levels, and hierarchies.** Now that your basic metadata server and cube information has been entered, you can define the different dimensions and their respective levels and hierarchies. You use the DIMENSION, HIERARCHY, and LEVEL statements here.

Note: For time-specific levels in a dimension, the LEVEL statement is required. Also, there can be only one time-specific dimension.

```
dimension products
   hierarchies=(products)
   caption="Products"
   sort_order=ascunformatted
   ;
hierarchy products
   levels=(product_category product_group product_id
   product_line product_name)
   ;
dimension date
   hierarchies=(date)
   caption="Date"
   type=time
   ;
hierarchy date
   levels=(year quarter month)
   ;
level year
   type=year
```

```
   ;
level quarter
   type=quarters
   ;
level month
   type=months
   ;
dimension geography
   hierarchies=(geography)
   caption="Geography"
   ;
hierarchy geography
   levels=(Country Region City)
   ;
dimension customer
   hierarchies=(customer)
   caption="Customer"
   ;
hierarchy customer
   levels=(customer_age customer_group customer_type)
   ;
dimension orders
   hierarchies=(orders)
   caption="Orders"
   ;
hierarchy orders
   levels=(order_date total_retail_price costprice_per_unit)
   ;
```

3 **Define measures.** You can now define the measures for the cube. A measure is an input column and a roll-up rule (statistic). Only certain measures are physically stored. Other measures are derived from the stored measures at run time. In this example, you want measures for the product CostPrice_per_unit average and a range.

You use the MEASURE statement here.

```
measure costprice_per_unitrange
   column=costprice_per_unit
   stat=range
   format=dollar10.2
   ;
measure costprice_per_unitavg
   column=costprice_per_unit
   stat=avg
   format=10.2
   ;
```

4 **Define member properties**. You can now define the member properties for any needed cube members. A member property is an attribute of a dimension member. A member property is also an optional cube feature that is created in a dimension to provide users with additional information about members. For this example, you can define the customer gender as a member property.

You use the PROPERTY statement here.

```
property gender
   column=customer_gender
   hierarchy=customer
```

```
   level=customer_id
   ;
```

5 **Define aggregations**. You can now define the aggregations for the cube. Aggregations are summaries of detailed data that is stored with a cube or referred by a cube. Their existence can reduce cube query time. If all aggregations are to be generated at the time of cube creation (MOLAP cube), then you can select aggregations in addition to the NWAY. The NWAY aggregation is the only aggregation that PROC OLAP creates by default.

You use the AGGREGATION statement here.

```
aggregation product_group product_line
   product_category country year
   name='ProductYear'
   ;
```

6 **Build the cube**. You can now build the cube. Execute the PROC OLAP statement within the SAS System or in batch-mode. Here is the complete PROC OLAP code.

```
proc olap data=olapsio.detail
   cube=/cubefolders/mycubefolder/Campaign1'
   path="c:\cubes"
   ;
metasvr host=localhost
   port=9999
   protocol=bridge
   userid=userid
   pw=pw
   repository=Foundation
   olap_schema="OLAP Schema"
   ;
dimension products
   hierarchies=(products)
   caption="Products"
   sort_order=ascunformatted
   ;
hierarchy products
   levels=(product_category product_group product_id
   product_line product_name)
   ;
dimension date
   hierarchies=(date)
   caption="Date"
   type=time
   ;
hierarchy date
   levels=(year quarter month)
   ;
level year
   type=year
   ;
level quarter
   type=quarters
   ;
level month
```

```
   type=months
   ;
dimension geography
   hierarchies=(geography)
   caption="Geography"
   ;
hierarchy geography
   levels=(Country Region City)
   ;
dimension customer
   hierarchies=(customer)
   caption="Customer"
   ;
hierarchy customer
   levels=(customer_age customer_group customer_type)
   ;
dimension orders
   hierarchies=(orders)
   caption="Orders"
   ;
hierarchy orders
   levels=(order_date total_retail_price costprice_per_unit)
   ;
measure costprice_per_unitrange
   column=costprice_per_unit
   stat=range
   format=dollar10.2
   ;
measure costprice_per_unitavg
   column=costprice_per_unit
   stat=avg
   format=10.2
   ;
property gender
   column=customer_gender
   hierarchy=customer
   level=customer_id
   ;
aggregation product_group product_line
   product_category country year
   name='ProductYear'
   ;
run;
```

Note: Any libraries must be specified before you run the PROC OLAP code.

Note: When a SAS OLAP cube is created, a directory for that cube is also created. This directory is assigned the same name as the cube, but in uppercase letters. For example, when you save a cube in `C:\olapcubes` and name the cube Campaigns, the cube is saved in the directory `C:\olapcubes\CAMPAIGNS`.

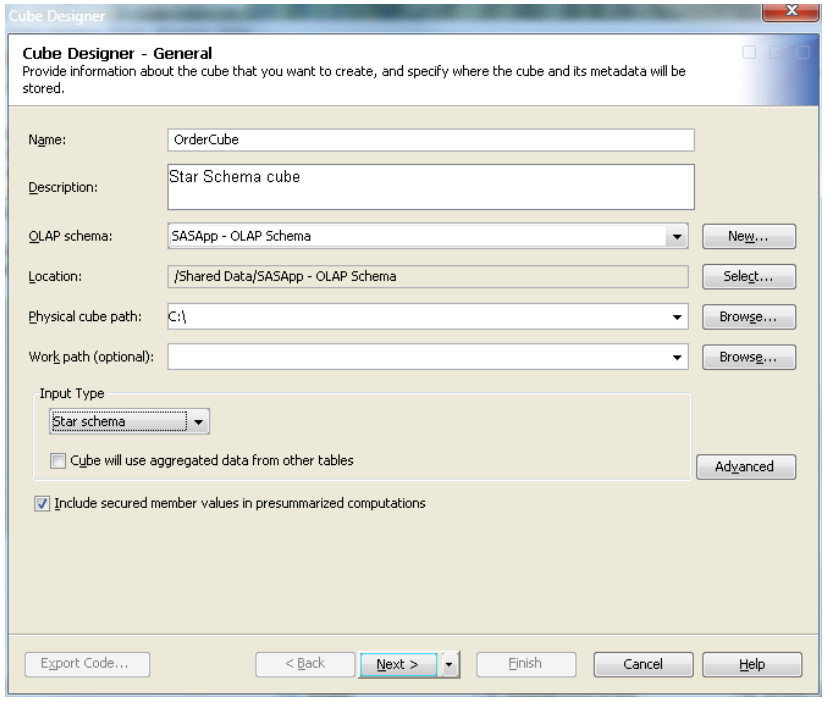
Building a Cube from a Star Schema

Overview

In this example, you build a cube from a star schema. A star schema is a data source that contains tables in a database in which a single fact table is connected to multiple dimension tables. With a cube based on a star schema, you identify the fact table, the dimension tables, and the keys that map the tables together. In this example, you use data from a product marketing campaign to establish measures and summaries of product statistics, geographic location of potential customers, and revenue.

Enter General Cube Information

After you have established a connection profile, you can begin to create a cube. Select **File** ⇨ **New** ⇨ **Cube**. On the Cube Designer – General page, enter the basic cube information. For the input type in this example, you click **Star Schema**. The following display shows fields that you enter information for.

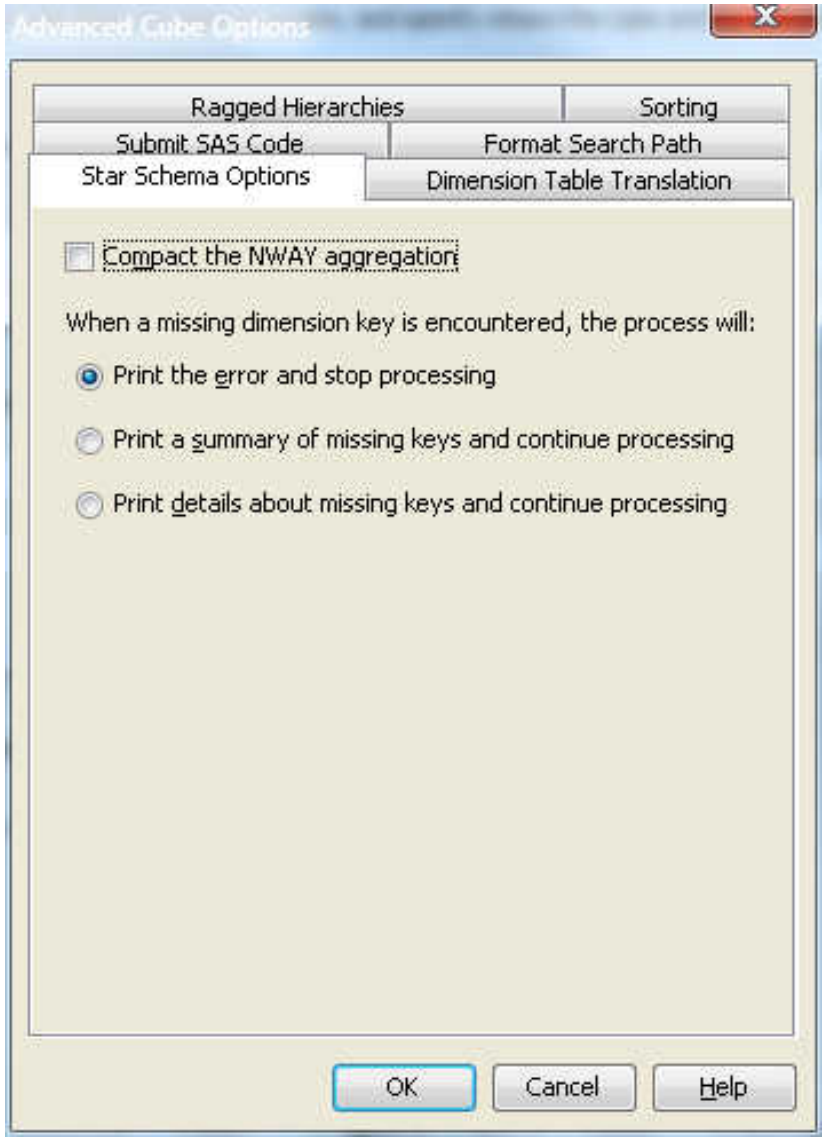


Enter information in the following fields:

- **Name**

- **Description**
- **OLAP schema**
- **Location** (SAS folder)
- **Physical path** (path in the file system to store the cube)
- **Work path** (path for temporary work files)
- **Input Type** (star schema)

After you have selected the star schema input type, you can click **Advanced** and define star schema options and dimension table translation languages for the cube. The following display shows the Advanced Cube Options dialog box. For star schema cubes you can specify options for missing keys and whether to compact the NWAY aggregation at build time.

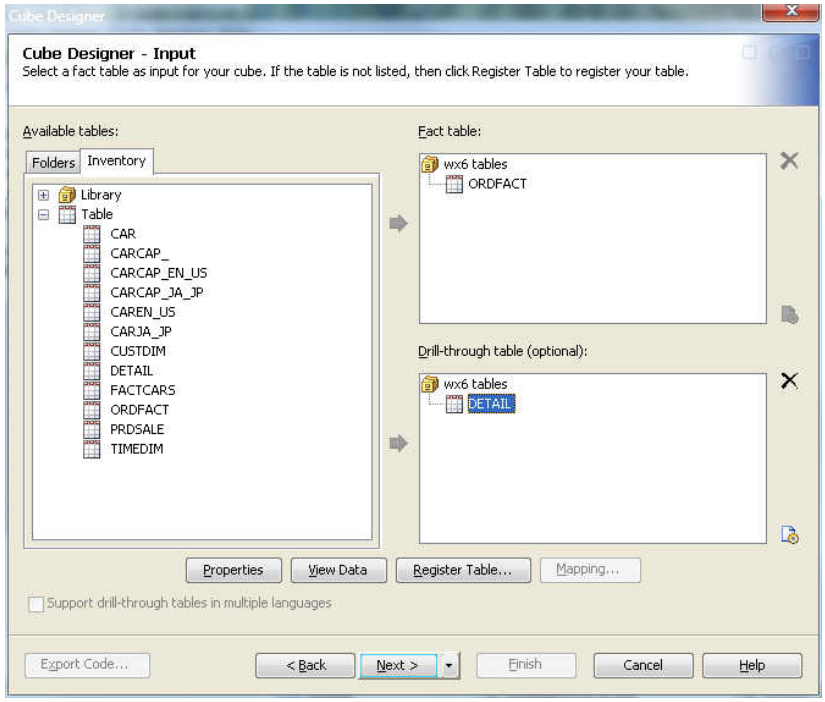


Select Fact and Dimension Tables

Overview

On the Cube Designer – Input page, select a fact table for your cube. If one does not exist for your data, click **Define Table**, and then define the source from which you will import your metadata. You can also select or define an optional drill-through table. Drill-through tables can be used by client applications to provide a view from processed data into the underlying data source. If a drill-through table does not exist for your data, click **Define Table**, and then define the source from which will import your metadata.

The following display shows the fact table ORDFACT that is selected for the example cube, along with the drill-through table DETAIL.



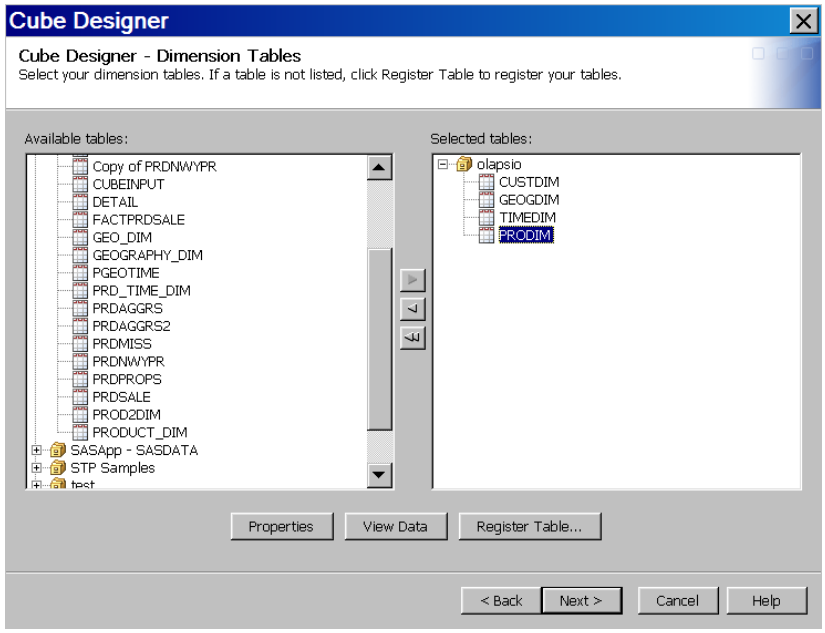
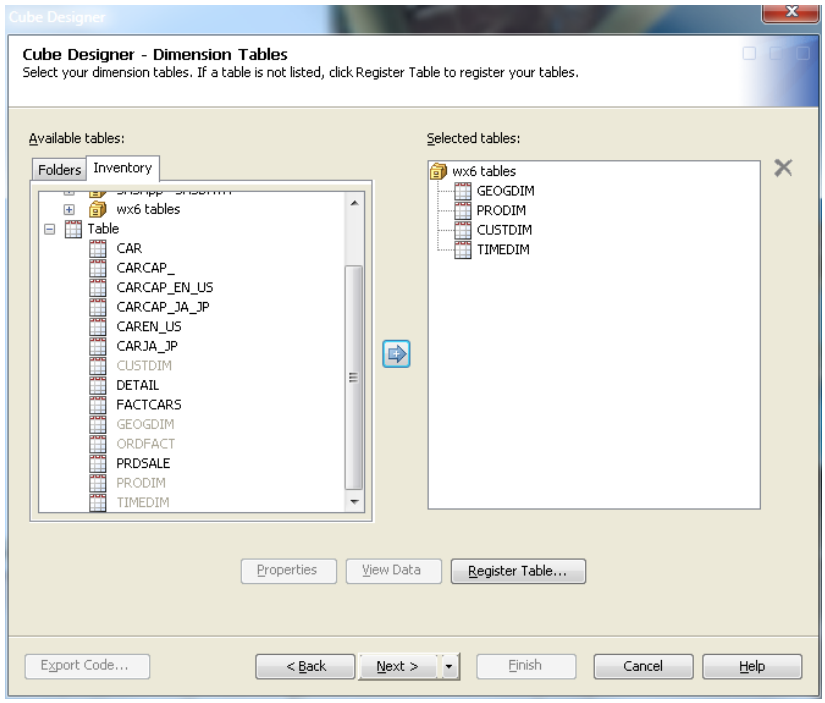


Table Options

The **Table Options** button is available in both the Cube Designer – Input and the Cube Designer – Drill-Through dialog boxes. It opens the Table Options dialog box. It enables you to specify data set options that are used to open the data set. For example, you could enter a WHERE clause or other information that subsets or filters the selected table when it is opened. The options are stored as part of the cube and then reapplied when the data is accessed at run time. You can also specify data set options in the Dimension Designer – General dialog box (for use with star schemas) and the Stored Aggregates dialog box (for use with summarized tables). For more information, see “Data Set Options” in the *SAS Language Reference: Concepts*.

Select Dimension Tables

After you choose a fact table, you select the dimension tables for the cube on the Cube Designer – Dimension Tables page. See the following display.

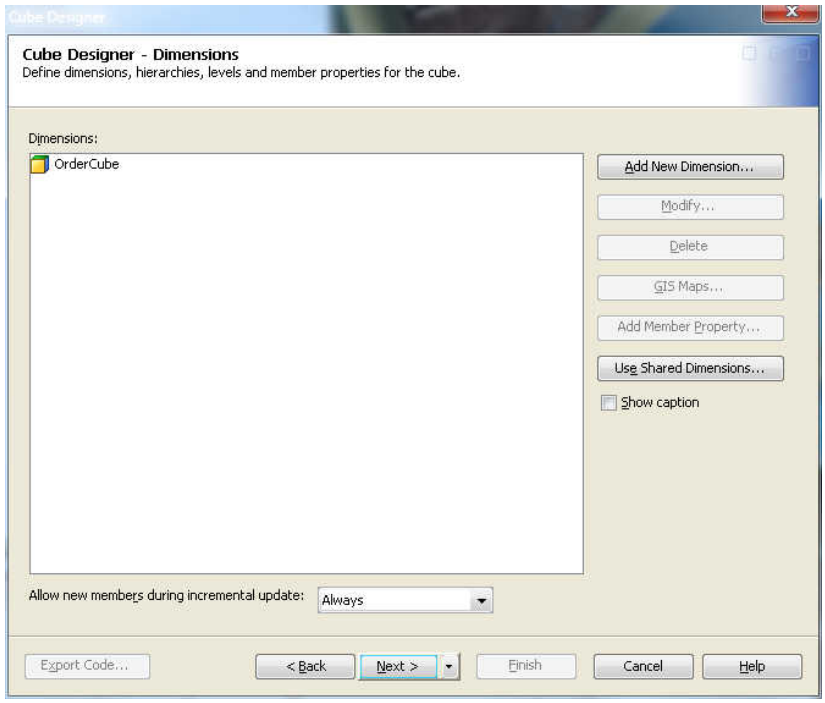


Define Dimensions, Levels, and Hierarchies

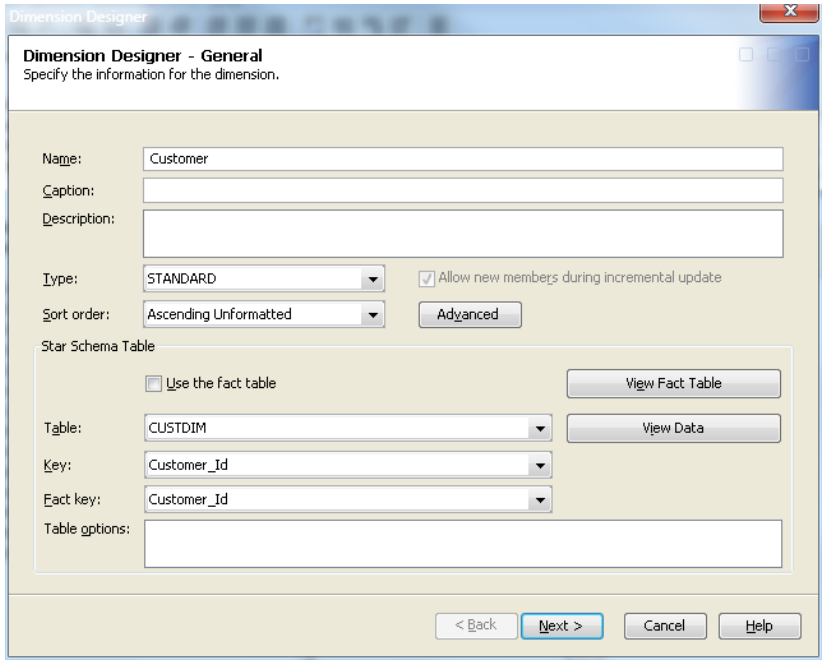
Now that your basic metadata server and cube information has been entered, you can define the different dimensions and their respective levels and hierarchies. For this example, the following dimensions are created:

- Product
- Time
- Geography
- Customer
- Organization

On the Cube Designer – Dimensions page, click **Add New Dimension**.



This opens the Dimension Designer – General page, as seen in the following display.



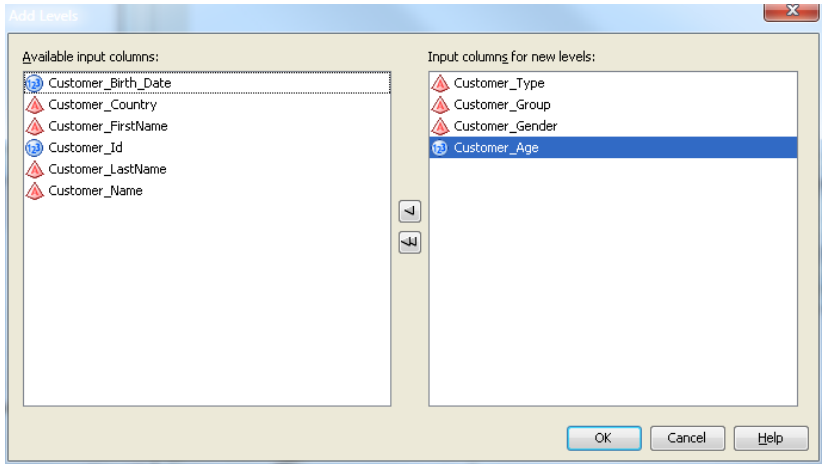
Enter information in the following fields:

- **Name**
- **Caption**
- **Description**
- **Type** (STANDARD, GEO, or TIME)
- **Sort order**

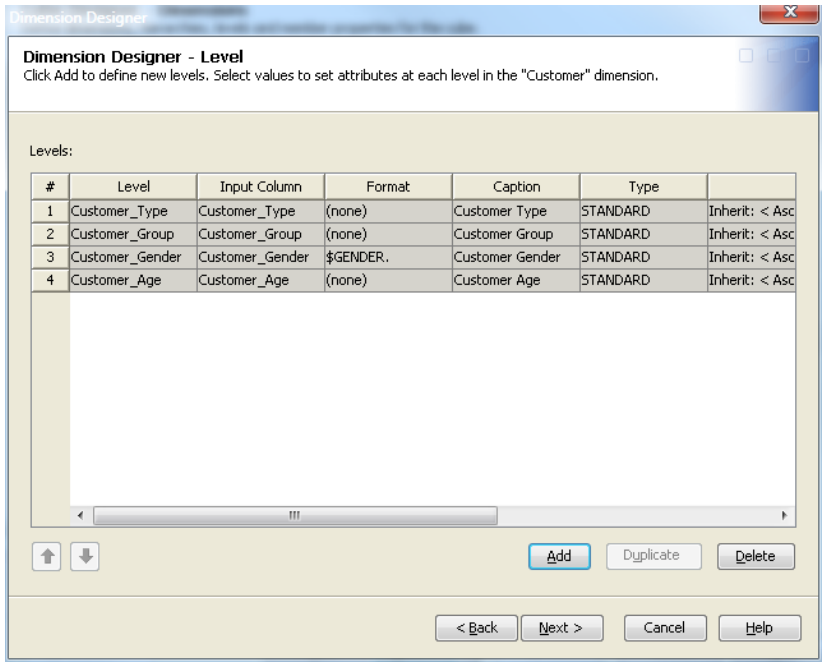
Because you are building a cube from star schema data, you must identify the table for the dimension, and the keys that link the dimension and fact tables together. For each dimension that you define, specify the following:

- **Table** (for the dimension that you are creating)
- **Key** (for the dimension that you are creating)
- **Fact key**
- any **Table options** for the selected dimension table

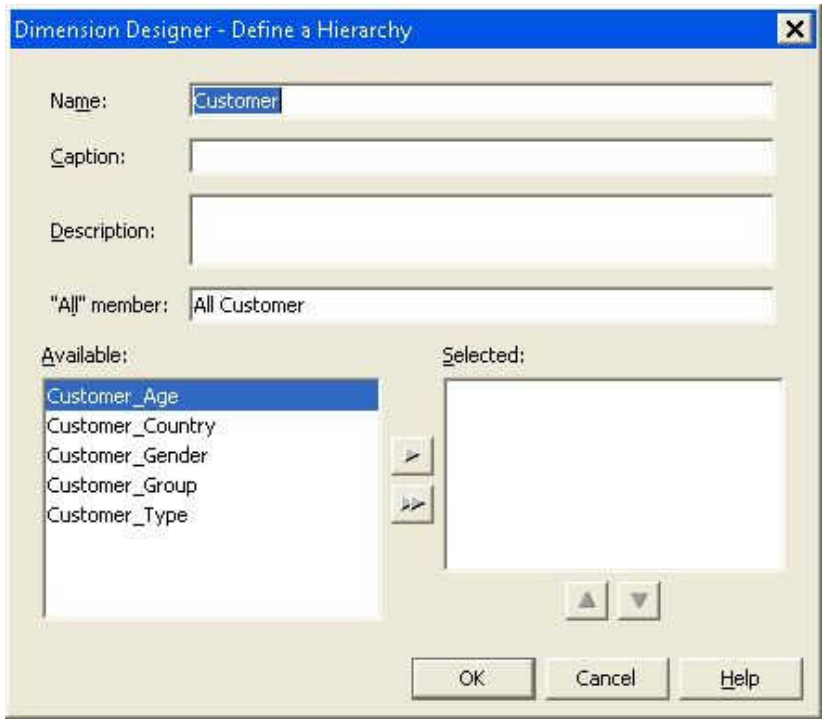
Select **Next**. This opens the Dimension Designer – Level page. Next, click **Add** to open the Add Levels page, as seen in the following display.



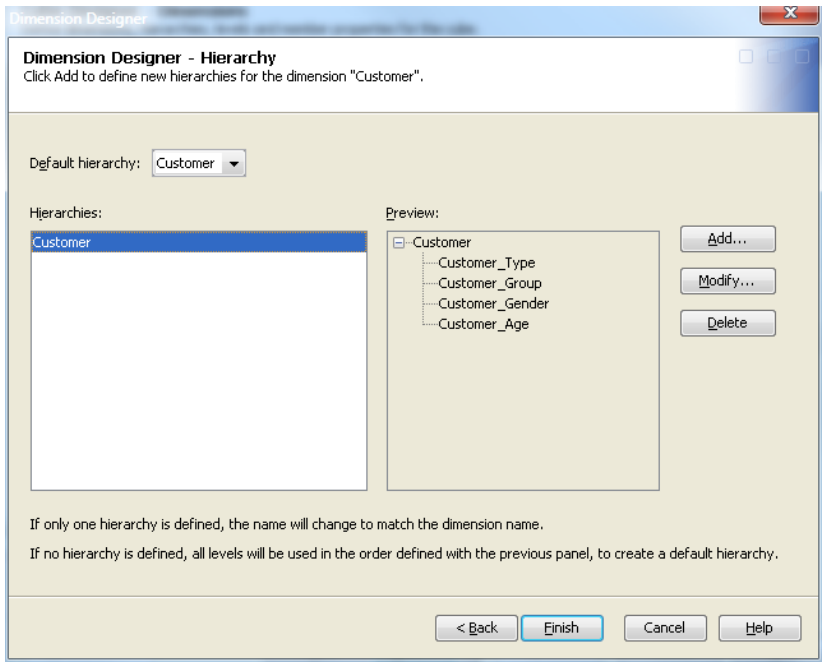
Select the levels that you want to add to the dimension. Select **OK** to return to the Dimension Designer – Level page, where the selected levels are listed. You can now define properties such as format, time type, and sort order for the levels that you have selected. See the following display.



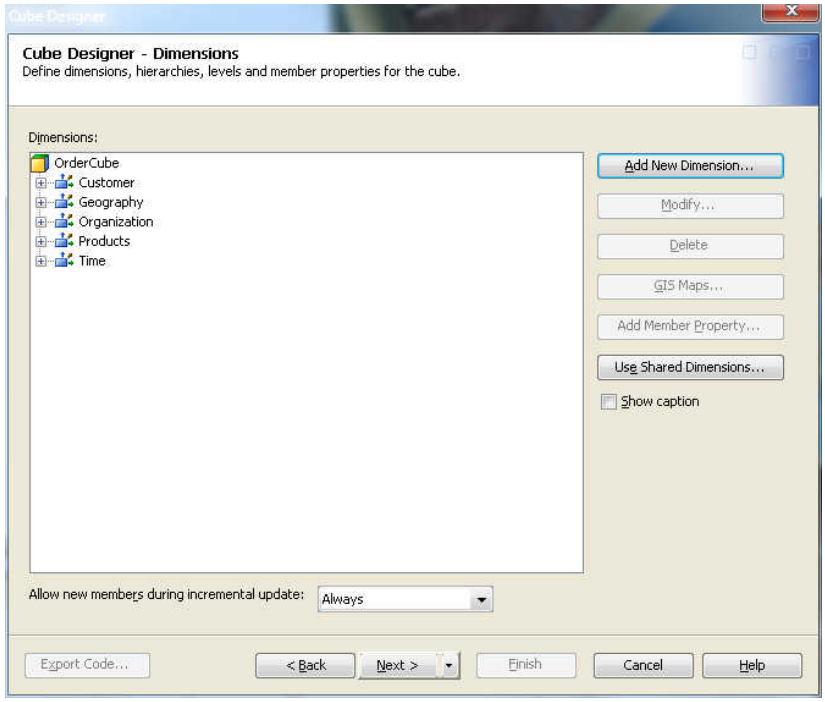
Next, define hierarchies for the levels on the Dimension Designer – Hierarchy page. Select **Add** to on the Dimension Designer – Define a Hierarchy page and individually select the levels for the hierarchy.



You can also click **Finish** on the Dimension Designer – Hierarchy page to accept the order of the levels that are defined on the previous Dimension Designer – Level page. If you select this option, the hierarchy is assigned the same name as the dimension. See the following display.



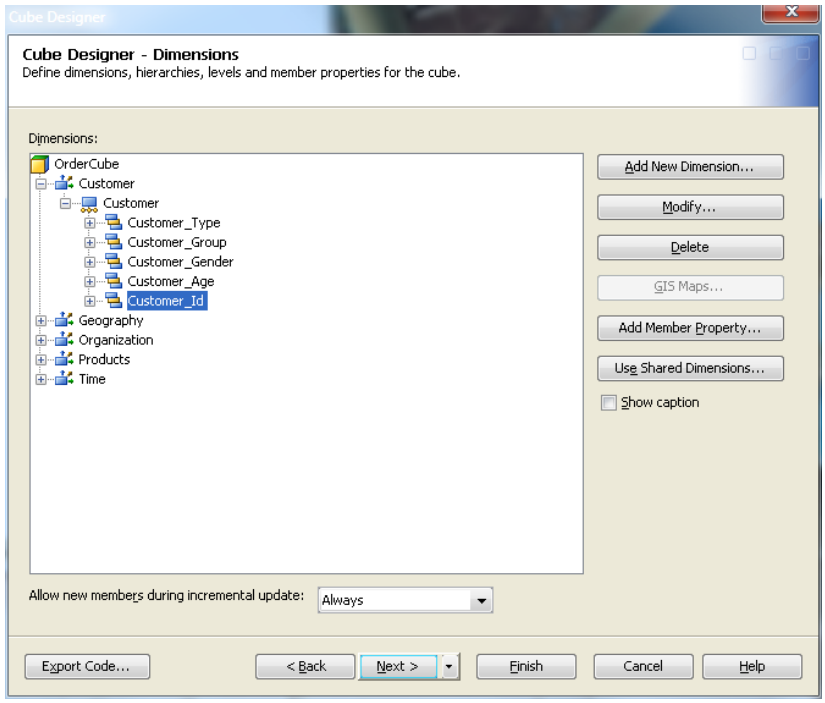
Repeat this process for each dimension. After you create each dimension, it is listed in the **Dimensions** panel of the Cube Designer – Dimensions page. See the following display.



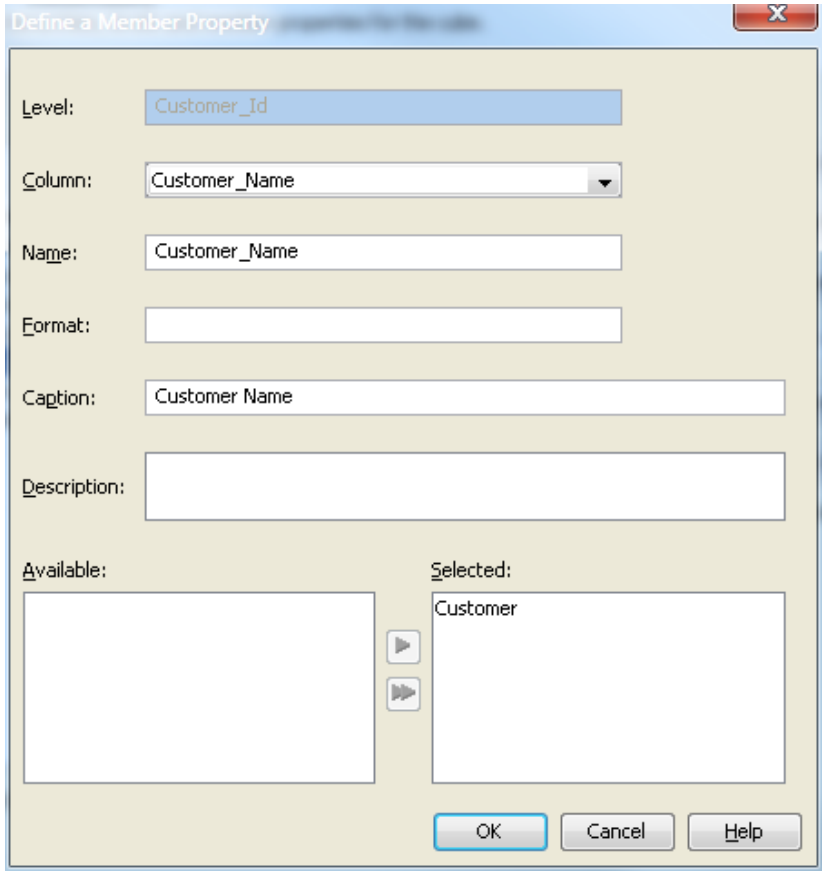
Note: If you are defining a TIME dimension, you can click **Supplied** on the Dimension Designer – Level page. See "Creating a TIME Dimension in SAS OLAP Cube Studio" on page 258 for more information.

Define Member Properties

You can now define the member properties for any needed cube members. A member property is an attribute of a dimension member. A member property is also an optional cube feature that is created in a dimension to provide users with additional information about members. For this example, you can define the customer name as a member property. On the Cube Designer – Dimensions page, select the level for the member property, and then click **Add Member Property**. See the following display.



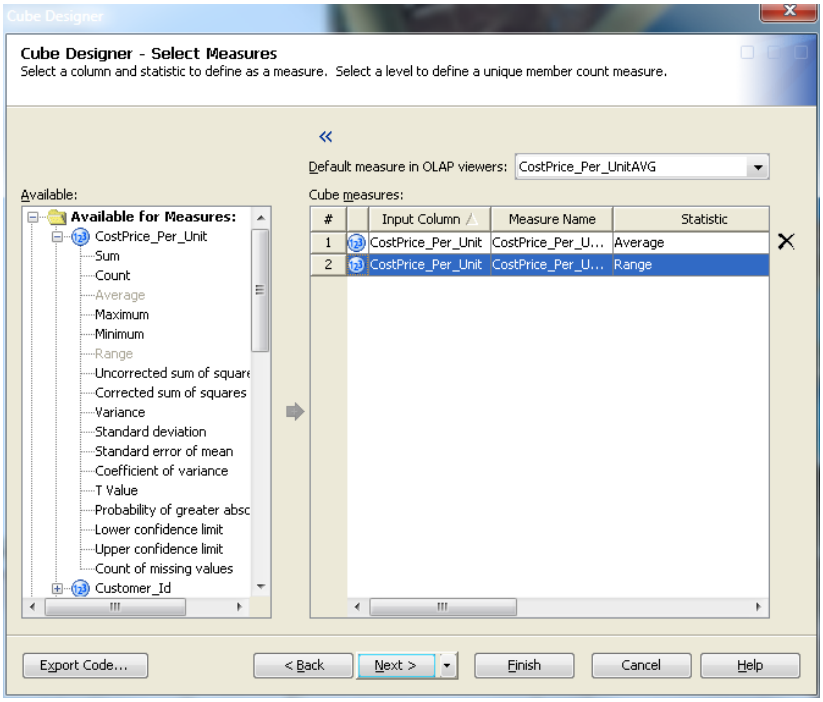
Define member properties in the Cube Designer – Member Property dialog box, as shown in the following display.



On the Define a Member Property dialog box, enter the member property name, column, format, and caption. Click **OK** to save your entries and define measures for your cube.

Define Measures

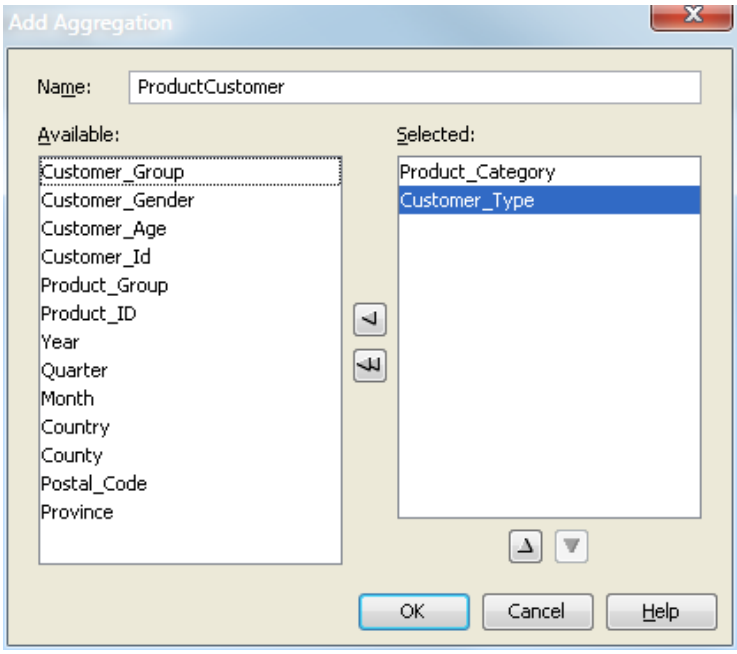
In this example, you define measures for the CostPrice Per Unit. Define the measures for the cube on the Cube Designer – Select Measures page, as shown in the following display.



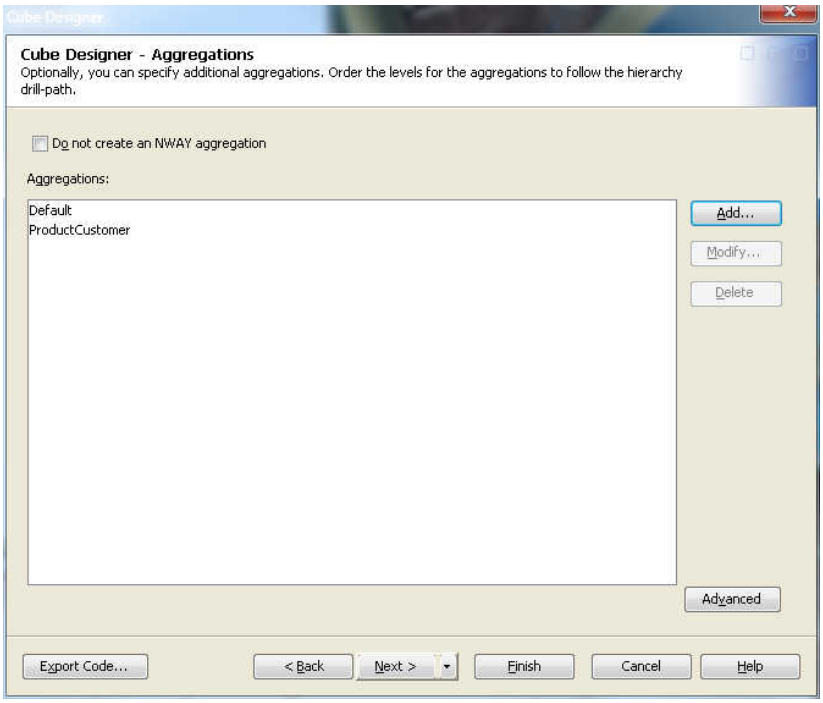
Define Aggregations

You can now define the aggregations for the cube. Aggregations are summaries of detailed data that are stored with a cube or referred to by a cube. They can help contribute to faster query response. You define the aggregations for the cube from the Cube Designer – Aggregations page.

Select **Add** to specify a user-defined aggregation. This opens the Add Aggregation dialog box, as shown in the following display. In this dialog box, you can select levels to add to your aggregation.



Select **OK** to return to the Cube Designer – Aggregations page, where the new aggregation is listed as shown in the following display.



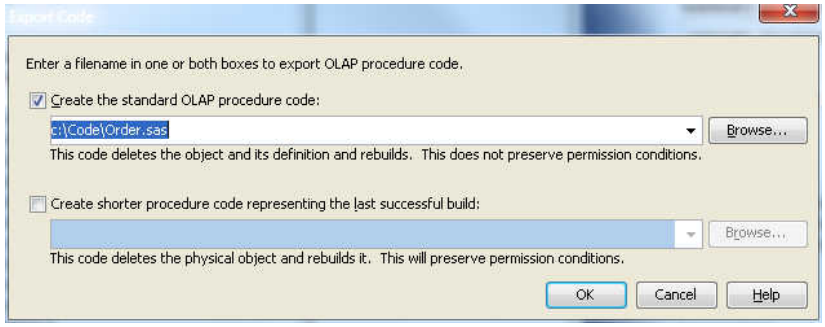
Select **Next** to go to the Cube Designer – Finish page.

Build the Cube

You can now build the cube. On the Cube Designer – Finish page, review the settings for the cube, and then click **Finish** to display the Finish dialog box. In the

Finish dialog box, you can choose to build the physical cube or save the cube definition and build the physical cube later.

You can also click **Export Code** to save the OLAP procedure code that is generated at the beginning of the build, as shown in the following display.



PROC OLAP CODE for the Star Schema Example

```
LIBNAME olapsio BASE "\\olap\tmp\libolap" ;

PROC OLAP
   CUBE                  = "/Shared Data/OLAPSchemas/OrderCube"
   PATH                  = 'c:\v9cubes'
   DESCRIPTION           = 'starschemacube'
   FACT                  = olapsio.ORDER_FACT
;


   METASVR
      HOST        = "J12345.na.abc.com"
      PORT        = 8561
      OLAP_SCHEMA = "SASApp - OLAP Schema";

   DIMENSION Customers
      CAPTION          = 'Customers'
      SORT_ORDER       = ASCENDING
      DIMTBL           = olapsio.CUSTDIM
      DIMKEY           = Customer_Id
      FACTKEY          = Customer_ID
      HIERARCHIES      = (
         Customers
         ) /* HIERARCHIES */;

      HIERARCHY Customers
         ALL_MEMBER = 'All Customers'
         CAPTION    = 'Customers'
         LEVELS     = (
            Customer_Type Customer_Group
            Customer_Gender Customer_Age
            ) /* LEVELS */
         DEFAULT;
```

```
   LEVEL Customer_Type
      CAPTION        =  'Customer Type'
      SORT_ORDER     =  ASCENDING;

   LEVEL Customer_Group
      CAPTION        =  'Customer Group'
      SORT_ORDER     =  ASCENDING;

   LEVEL Customer_Gender
      FORMAT         =  $GENDER.
      CAPTION        =  'Customer Gender'
      SORT_ORDER     =  ASCENDING;

   LEVEL Customer_Age
      CAPTION        =  'Customer Age'
      SORT_ORDER     =  ASCENDING;

DIMENSION Geography
   CAPTION          = 'Geography'
   TYPE             = GEO
   SORT_ORDER       = ASCENDING
   DIMTBL           = olapsio.GEOGDIM
   DIMKEY           = Street_Id
   FACTKEY          = Street_ID
   HIERARCHIES      = (
      Geography
      ) /* HIERARCHIES */;

   HIERARCHY Geography
      ALL_MEMBER = 'All Geography'
      CAPTION    = 'Geography'
      LEVELS     = (
         Country Region State
         ) /* LEVELS */
      DEFAULT;

   LEVEL Country
      FORMAT         =  $COUNTRY.
      CAPTION        =  'Country'
      SORT_ORDER     =  ASCENDING;

   LEVEL Region
      CAPTION        =  'Region Name'
      SORT_ORDER     =  ASCENDING;

   LEVEL State
      CAPTION        =  'State'
      SORT_ORDER     =  ASCENDING;

DIMENSION Organization
   CAPTION          = 'Organization'
   SORT_ORDER       = ASCENDING
   DIMTBL           = olapsio.ORGDIM
   DIMKEY           = Employee_Id
   FACTKEY          = Employee_ID
   HIERARCHIES      = (
```

```
      Organization
      ) /* HIERARCHIES */;

   HIERARCHY Organization
      ALL_MEMBER = 'All Organization'
      CAPTION    = 'Organization'
      LEVELS     = (
         Company Group Department
         ) /* LEVELS */
      DEFAULT;

   LEVEL Company
      CAPTION        =  'Company'
      SORT_ORDER     =  ASCENDING;

   LEVEL Group
      CAPTION        =  'Group'
      SORT_ORDER     =  ASCENDING;

   LEVEL Department
      CAPTION        =  'Department'
      SORT_ORDER     =  ASCENDING;

DIMENSION Product
   CAPTION          = 'Product'
   SORT_ORDER       = ASCENDING
   DIMTBL           = olapsio.PRODIM
   DIMKEY           = Product_ID
   FACTKEY          = Product_ID
   HIERARCHIES      = (
      Product
      ) /* HIERARCHIES */;

   HIERARCHY Product
      ALL_MEMBER = 'All Product'
      CAPTION    = 'Product'
      LEVELS     = (
         Product_Category Product_Group Product_Line
         ) /* LEVELS */
      DEFAULT;

   LEVEL Product_Category
      CAPTION        =  'Product Category'
      SORT_ORDER     =  ASCENDING;

   LEVEL Product_Group
      CAPTION        =  'Product Group'
      SORT_ORDER     =  ASCENDING;

   LEVEL Product_Line
      CAPTION        =  'Product Line'
      SORT_ORDER     =  ASCENDING;

DIMENSION Time
   CAPTION          = 'Time'
   TYPE             = TIME
```

```
   SORT_ORDER       = ASCFORMATTED
   DIMTBL           = olapsio.TIMEDIM
   DIMKEY           = Date
   FACTKEY          = Order_Date
   HIERARCHIES      = (
      Time
      ) /* HIERARCHIES */;

   HIERARCHY Time
      ALL_MEMBER = 'null'
      LEVELS     = (
         Year Quarter Month
         ) /* LEVELS */
      DEFAULT;

   LEVEL Year
      COLUMN         =  Date
      FORMAT         =  YEAR4.
      TYPE           =  YEAR
      CAPTION        =  'Year'
      SORT_ORDER     =  ASCFORMATTED;

   LEVEL Quarter
      COLUMN         =  Date
      FORMAT         =  QTR1.
      TYPE           =  QUARTERS
      CAPTION        =  'Quarter'
      SORT_ORDER     =  ASCFORMATTED;

   LEVEL Month
      COLUMN         =  Date
      FORMAT         =  MONNAME9.
      TYPE           =  MONTHS
      CAPTION        =  'Month'
      SORT_ORDER     =  ASCFORMATTED;

PROPERTY Ages
   LEVEL          = Customer_Age
   COLUMN         = Customer_Birth_Date
   CAPTION        = 'Customer Birth Date'
   HIERARCHY      = (
      Customers
      ) /* HIERARCHIES */;

MEASURE Total_Retail_PriceMAX
   STAT        = MAX
   COLUMN      = Total_Retail_Price
   CAPTION     = 'Maximum Total_Retail_Price'
   FORMAT      = DOLLAR13.2
   DEFAULT;

MEASURE Total_Retail_PriceAVG
   STAT        = AVG
   COLUMN      = Total_Retail_Price
   CAPTION     = 'Average Total_Retail_Price'
   FORMAT      = DOLLAR13.2;
```

```
MEASURE CostPrice_Per_UnitMAX
   STAT        = MAX
   COLUMN      = CostPrice_Per_Unit
   CAPTION     = 'Maximum CostPrice_Per_Unit'
   FORMAT      = DOLLAR13.2;

MEASURE CostPrice_Per_UnitAVG
   STAT        = AVG
   COLUMN      = CostPrice_Per_Unit
   CAPTION     = 'Average CostPrice_Per_Unit'
   FORMAT      = DOLLAR13.2;

AGGREGATION /* Default */
   /* levels */
   Company Country Customer_Age Customer_Gender Customer_Group
   Customer_Type Department Group Month Product_Category
   Product_Group Product_Line Quarter
   Region State Year
   / /* options */
   NAME      = 'Default';

AGGREGATION /* ProductCustomer */
   /* levels */
   Customer_Type Customer_Group Customer_Gender
   Customer_Age Product_Category
   / /* options */
   NAME      = 'ProductCustomer';

FORMAT Customer_Birth_Date DATE9.;

RUN;
```

PROC OLAP Statements and Options for a Star Schema

The following table lists the PROC OLAP statements and options that you use to load a cube from a star schema. A star schema is a set of input tables that are defined in a repository. The set of tables includes a single fact table and one or more dimension tables. The fact table must contain at least one numeric analysis column for each set of measures that is generated. To specify the data source for a star schema, you must use the FACT=, DIMTBL=, DIMKEY=, and FACTKEY= options.

Table 9.2 *Statements and Options Used to Load Cubes from a Star Schema*

Statements	Options	Required or Optional
PROC OLAP	FACT=	Required
	CUBE=	Required

Statements	Options	Required or Optional
	PATH=	Required
	DESC=	Optional
	NO_NWAY	Optional
	WORKPATH=	Optional
	DT_TABLE	Optional
METASVR	OLAP_SCHEMA=	Required
	REPOSITORY=	Optional
	HOST=	Optional
	PORT=	Optional
	PROTOCOL=	Optional
	USERID=	Optional
	PW=	Optional
DIMENSION	HIERARCHIES=	Required
	DIMTBL=	Required for cubes that support one locale. If the cube will contain multiple national languages, replace this option with DIMTABLELIBREF= and DIMTABLEMEMPREF=.
	FACTKEY=	Required
	DIMKEY=	Required
	DIMTABLELIBREF	Required if you build a cube that will contain multiple national languages. Replaces DIMTBL=.
	DIMTABLEMEMPREF	Required if you build a cube that will contain multiple national languages. Replaces DIMTBL=.
	DESC=	Optional
	CAPTION=	Optional
	TYPE=TIME	Required only for TIME dimensions

Statements	Options	Required or Optional
	SORT_ORDER=	Optional
LEVEL		The LEVEL statement is optional unless you want to specify time periods for each level in a TIME dimension. If you specify a time period for one level, then you must specify a time period for all levels. To specify a time period, you use the TYPE= option.
HIERARCHY	LEVELS=	Required
	DESC=	Optional
	CAPTION=	Optional
MEASURE	STAT=	Required
	COLUMN | ANALYSIS=	Required
	AGGR_COLUMN=	Required if you use the AGGREGATION statement with the TABLE= option
	DESC=	Optional
	CAPTION=	Optional
	UNITS=	Optional
	FORMAT=	Optional
	DEFAULT=	Optional
PROPERTY	prop-name	Required
	LEVEL=	Required
AGGREGATION		The AGGREGATION statement is optional unless you are creating additional aggregations. In that case, you must specify the names of the contiguous levels to be used to create the aggregation. Use the TABLE= option for cubes that contain aggregated data from tables other than the input data source.

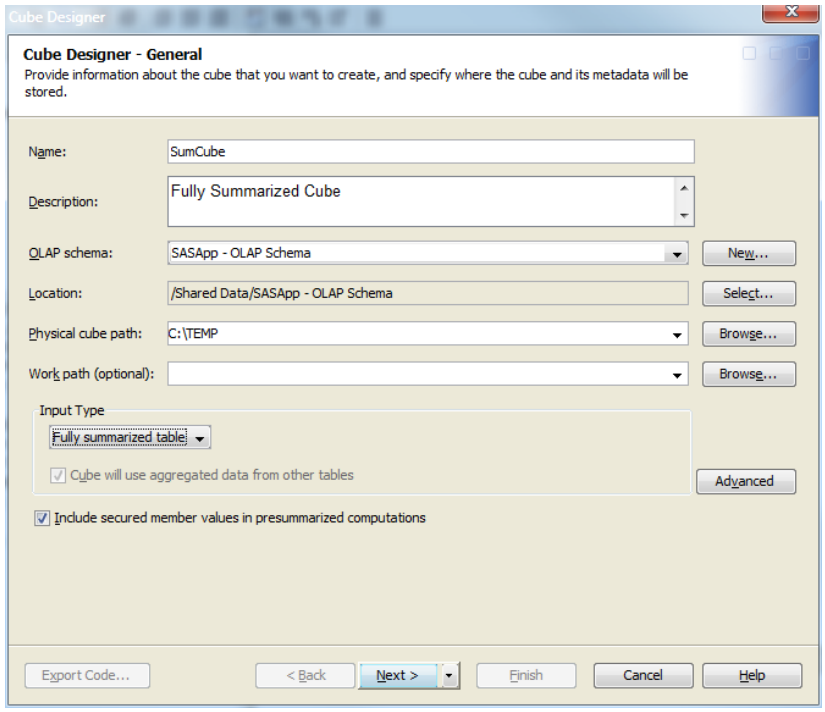
Building a Cube from a Summary Table

Overview

In this example, you build a cube with fully summarized data. A summary table is a data source that contains a crossing of all dimensions for a cube. In this example, the summary table PRDNWYPR contains sales data from a furniture company. The table contains columns for location, sales, date, and product values. It also contains stored measures for actual and predicted sales figures.

Enter General Cube Information

After you have established a connection profile, you can begin to create a cube. Select **File** ⇨ **New** ⇨ **Cube**. On the Cube Designer – General page, enter the basic cube information. For this example input type, you click **Fully Summarized Table**. The following display shows fields that you enter information for.



- Enter information in the following fields:
  - **Name**
  - **Description**

- **OLAP schema**
- **Location** (SAS folder)
- **Physical cube path** (path in the file system to store the cube)
- **Work path** (path for temporary work files)
- **Input Type** (fully summarized table)

Note: When the **Fully summarized table** input type is selected, the option **Cube will use aggregated data from other tables** is automatically selected.

Select an Input Table

Overview

On the Cube Designer – Input page, select an input table for your cube. If one does not exist for your data, click **Register Table**, and then define the source that you will import your metadata from.

Drill-Through Table

In the Cube Designer – Input page, you can also select or define an optional drill-through table. Drill-through tables can be used by client applications to provide a view from processed data into the underlying source data.

If a drill-through table does not exist for your data, click **Register Table**.

The following display shows how the summarized table PRDNWYPR is selected for the example cube. No drill-through table has been selected.

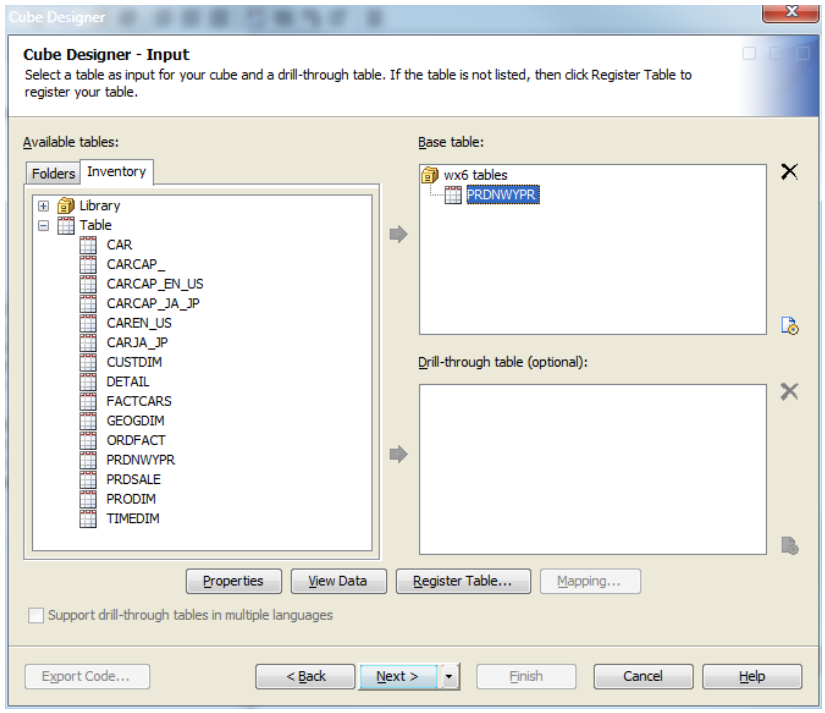


Table Options

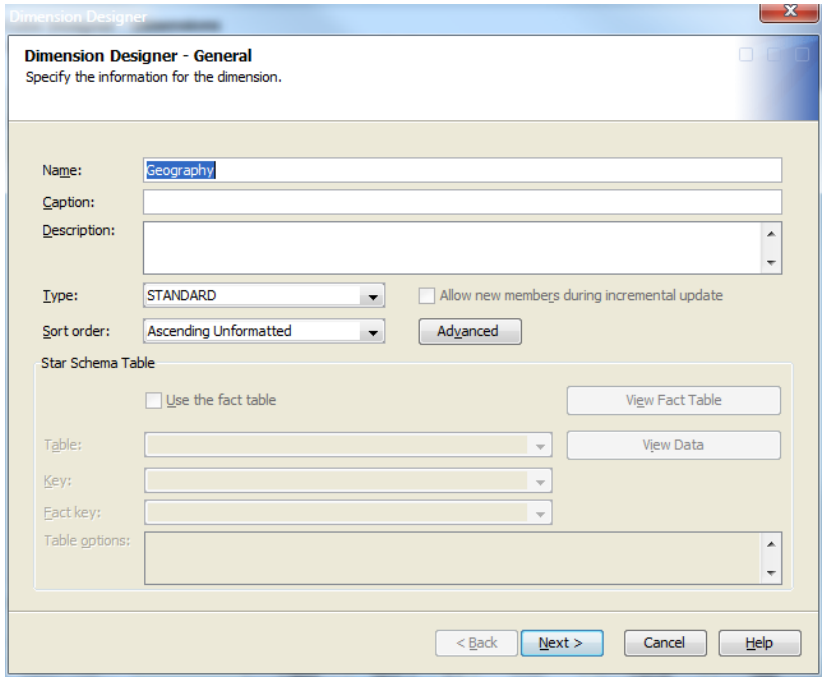
On the Cube Designer – Input page, you can specify data set options that are used to open either the detail table or drill-through table for your cube. The  button near the **Base table** tree or the **Drill-thru table** tree is available in both the Cube Designer – Input and the Cube Designer – Drill-Through dialog boxes. The button opens the Table Options dialog box. It enables you to specify data set options that are used to open the data set. For example, you could enter a WHERE clause or subsetting information that is then applied to the selected table when it is opened. The options are stored as part of the cube and then reapplied when the data is accessed at run time. You can also specify data set options in the Dimension Designer – General dialog box (for use with star schemas) and the Stored Aggregates dialog box (for use with summarized tables). For more information, see “Data Set Options” in the *SAS Language Reference: Concepts*.

Define Dimensions, Levels, and Hierarchies

Now that your basic metadata server and cube information has been entered, you can define the different dimensions and their respective levels and hierarchies. For this example, the following dimensions are created:

- Products
- Dates
- Geography

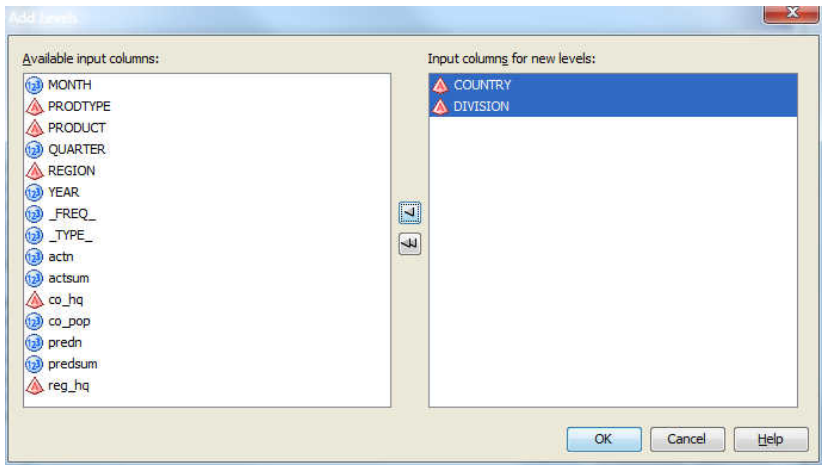
On the Cube Designer – Dimensions page, click **Add New Dimension**. This displays the Dimension Designer – General page, as shown in the following display.



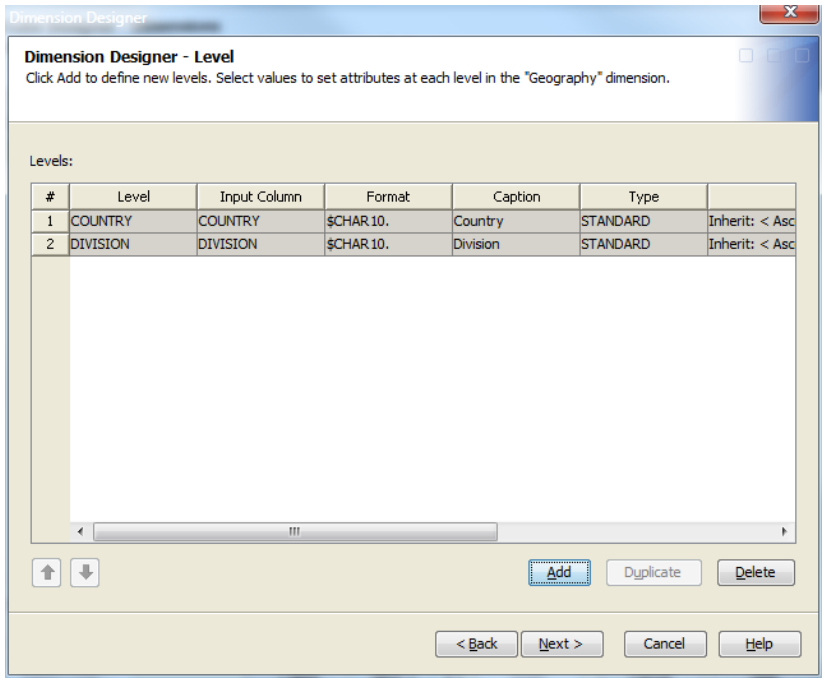
Enter information in the following fields:

- **Name**
- **Caption**
- **Description**
- **Type** (Standard, GEO, or TIME)
- **Sort order**

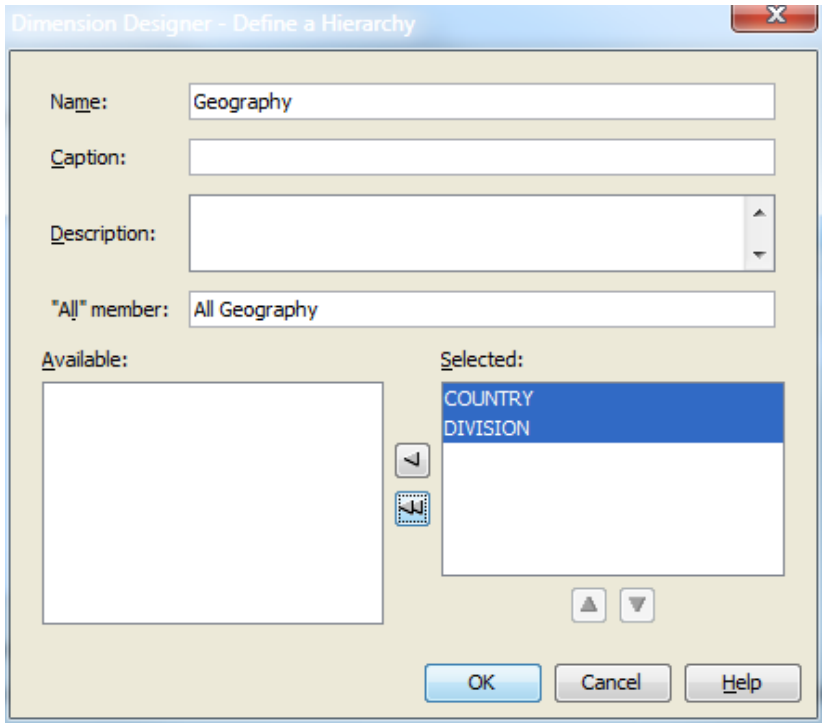
Select **Next**. This opens the Dimension Designer – Level page. Next, click **Add** to open the Add Levels dialog box, as shown in the following display.



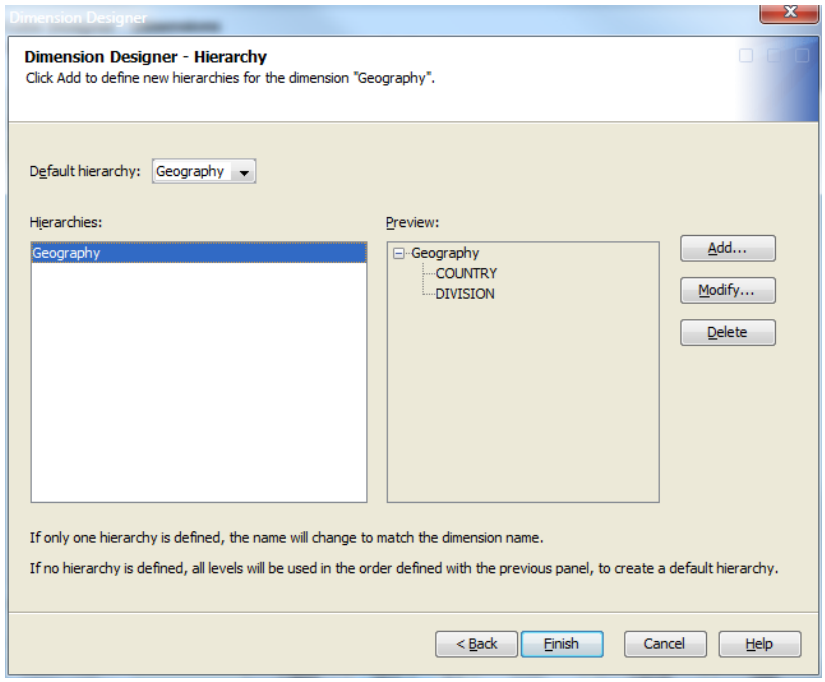
Select the levels that you want to add to the dimension. Select **OK** to return to the Level page, where the selected levels are listed. You can now define properties such as format, time type, and sort order for the levels that you have selected. See the following display.



Next, define hierarchies for the levels on the Dimension Designer – Hierarchy page. You can click **Add** to open the Define a Hierarchy page and individually select the levels for the hierarchy.



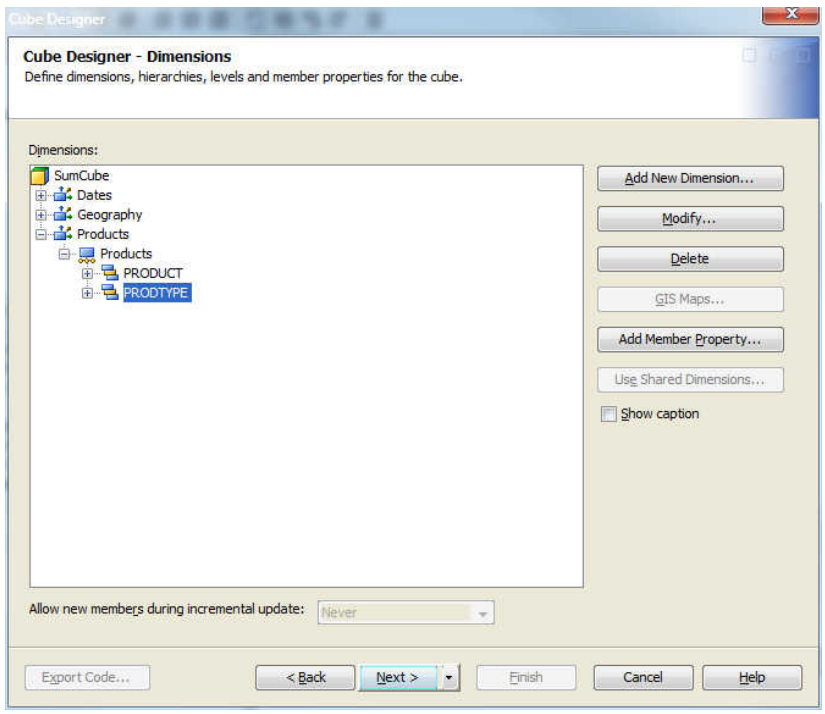
Or you can click **Finish** to accept the order of the levels that are defined on the previous Dimension Designer – Level page. If you select this option, the hierarchy is assigned the same name as the dimension. See the following display.



Repeat this process for each dimension. For this example, the following dimensions are created:

- Geography: (levels: Country, Division)
- Products: (levels: Product, Product Type)
- Dates: (Year, Quarter, Month)

After you create each dimension it is listed in the **Dimensions** panel of the Cube Designer – Dimensions page. See the following display:



Define Member Properties

You can now define the member properties for any needed cube members. A member property is an attribute of a dimension member. A member property is also an optional cube feature that is created in a dimension to provide users with additional information about members. Define member properties in the Cube Designer – Dimensions page by selecting a level, and then selecting **Add Member Property**.

Define Measures

Overview

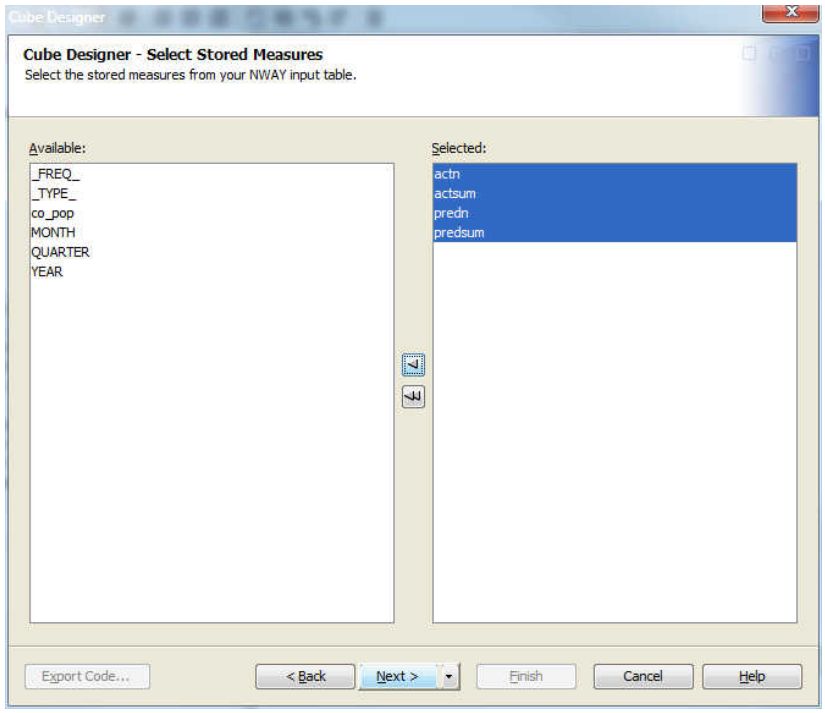
For this example, stored measures and derived measures are created. Stored measures are base measures that are loaded from the fully summarized table. When you are creating a cube from a fully summarized table, the table must have one column for each stored measure in your cube. The base statistics are SUM, N, MIN, MAX, NMISS, and USS.

Derived measures are measures that are built from the stored measures that you have selected for the cube. Derived measures are assigned to an analysis group when they are created. An analysis group is used to identify the numeric column in the original unsummarized data source that was used as the analysis variable for the stored measure. It can also be a name that identifies a logical association between several stored measures.

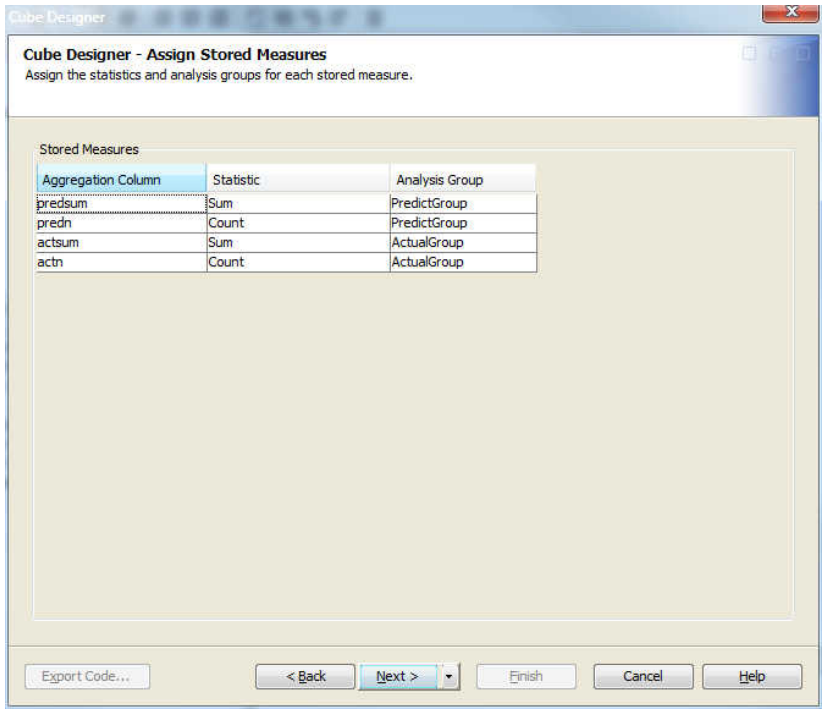
Note: For further information about measure statistics, see “Statistics Available for Measures ” on page 78 .

Stored Measures

You can now select the stored (base) measures for the cube in the Cube Designer – Select Stored Measures page. From the list of available measures, select the stored measures that you want to include in the cube. For this example **actn**, **actsum**, **predn**, and **predsum** are included.

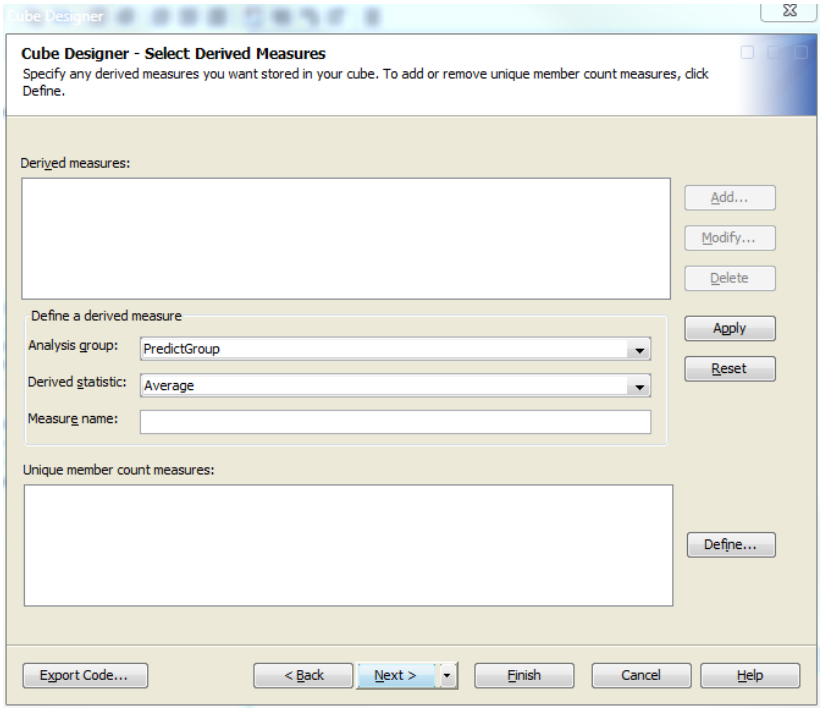


On the Cube Designer – Assign Stored Measures page, you can specify the **Statistic** and **Analysis Group** options for the stored measures. For the **Statistic**, select the appropriate statistic from the drop-down menu. For the **Analysis Group**, enter a name that identifies a logical association between the stored measures. The **Analysis Group** name can identify the numeric column in the original unsummarized data source that was used as the analysis variable for the stored measure. If the table contained two measures from the same analysis column, both of the base measures should have the same analysis group specified.

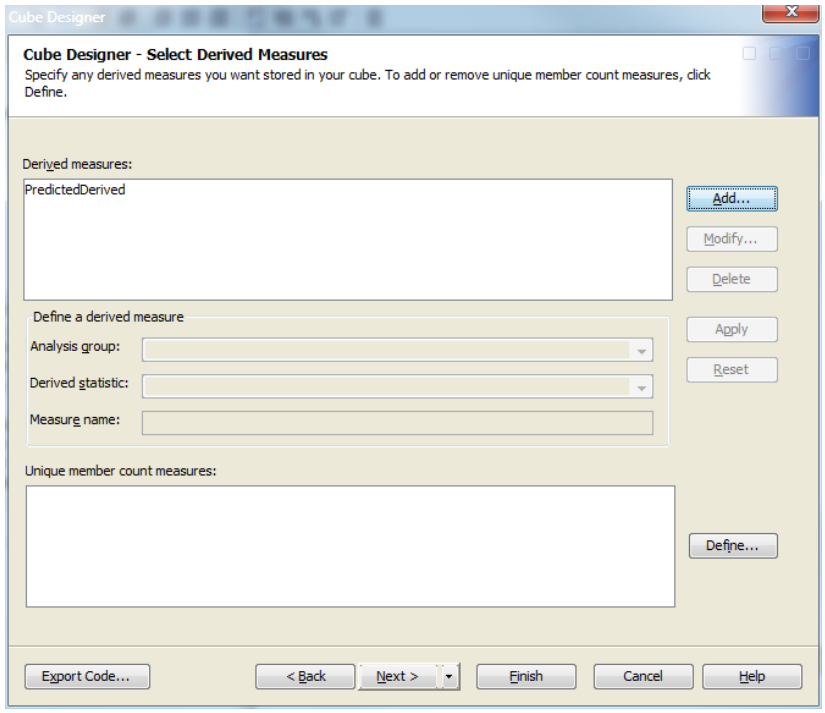


Derived Measures

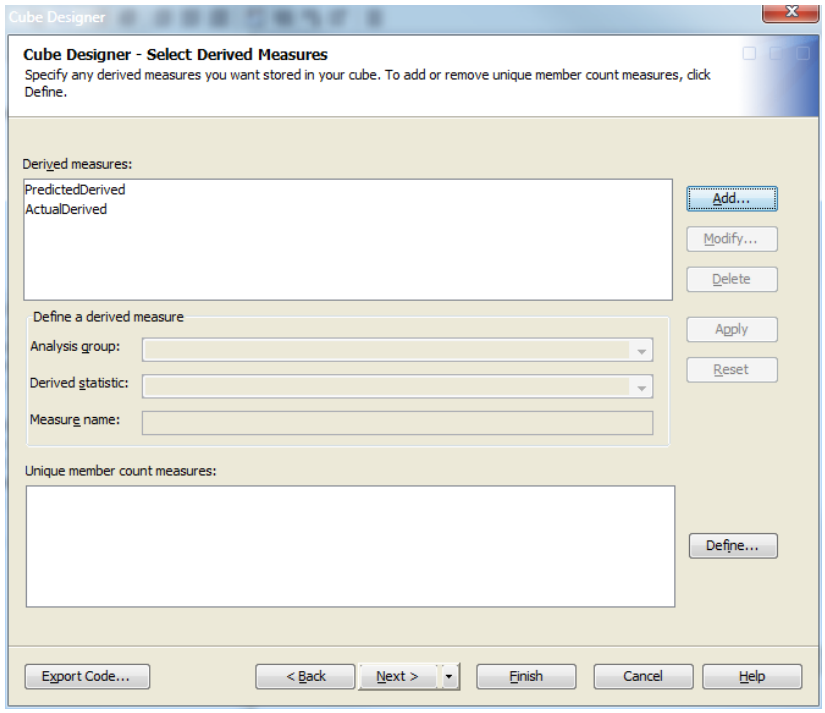
On the Cube Designer – Select Derived Measures page, specify the measures that are derived from the stored measures. Each derived measure is based on a set of required stored measures. If the stored measures for an analysis group do not include all those required for a specific derived measure, then that measure cannot be included in the cube. On the **Define a derived measure** panel, select the **Analysis group**, **Derived statistic**, and **Measure name** for the derived measure that you are creating.



When you are finished, click **Apply**. The derived measure is listed in the **Derived Measures** list.

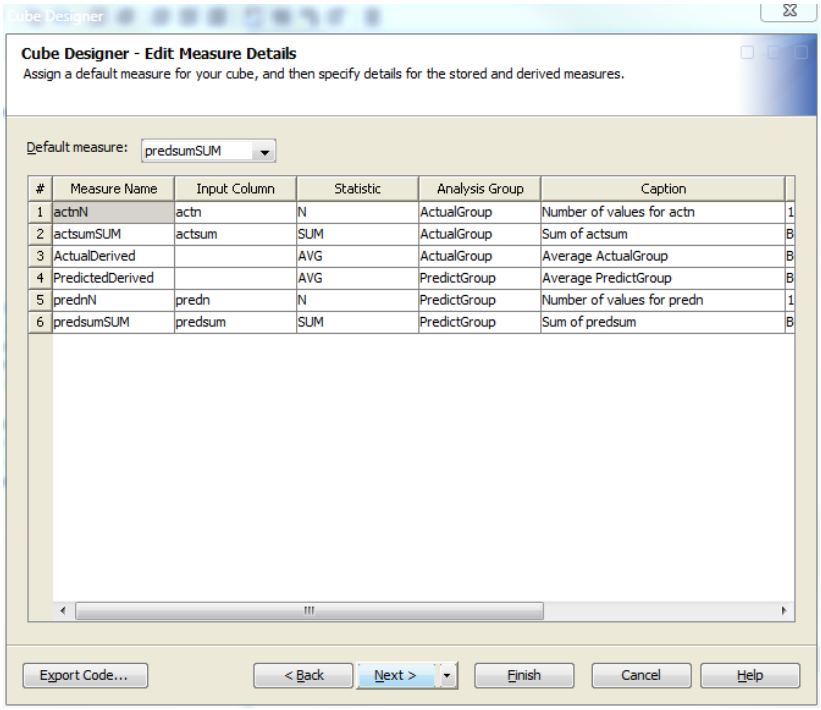


For this example, two derived measures are created: ActualDerived and Predicted Derived.



Edit Measure Details

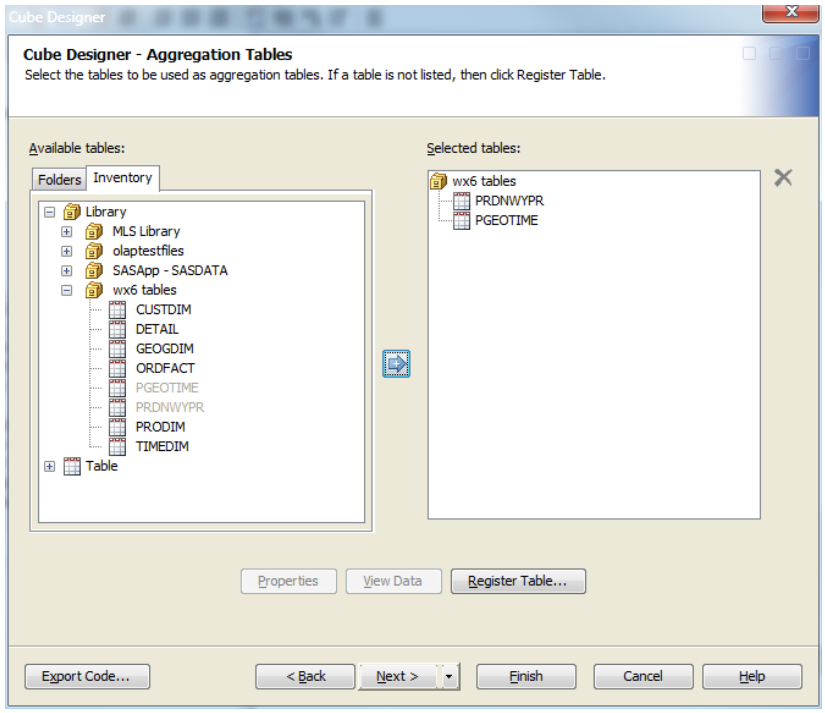
On the Cube Designer – Edit Measure Details page, you can select a default measure and modify measure details for the stored and derived measures.



Define Aggregation Tables

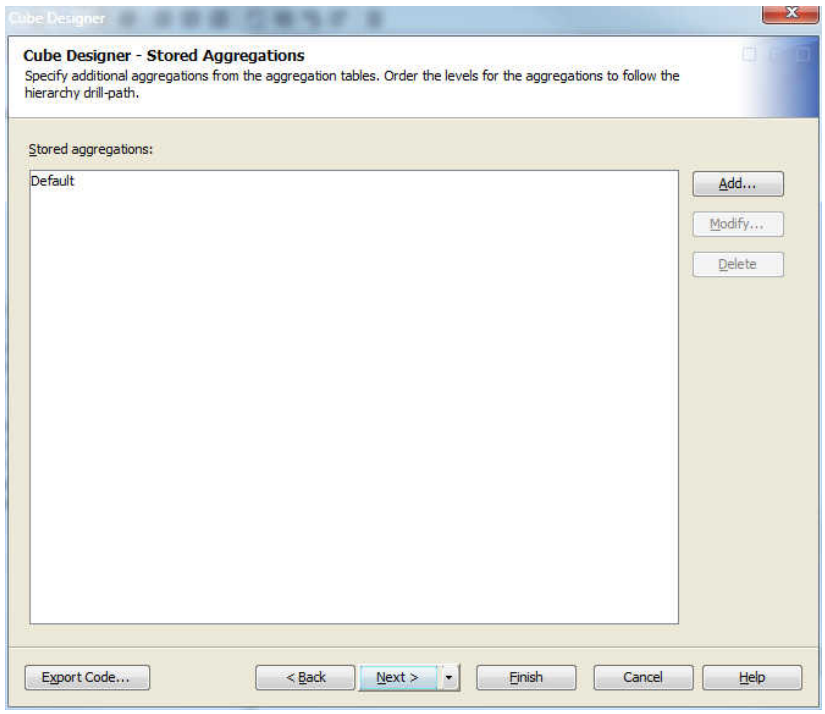
On the Cube Designer – Aggregation Tables page, you associate aggregation tables with the summarized data source that you specified as the input data source for the cube. When you open the Cube Designer – Aggregation Tables page, the input data table that you selected to build your cube with is listed in the **Selected tables** list. You can then select a table to use as the aggregation table from the **Available tables** list and move it to the **Selected tables** list. For this example, the table PGEOTIME is used as the aggregation table.

Note: If the cube is loaded from a fully summarized data source, then the measure names within the selected aggregation tables must match the measure names in the input data source. If the cube is loaded from a detail table or a star schema, then all of the selected aggregation tables must use the same measure names. For all cubes, the levels must be the same as those in the input data source.

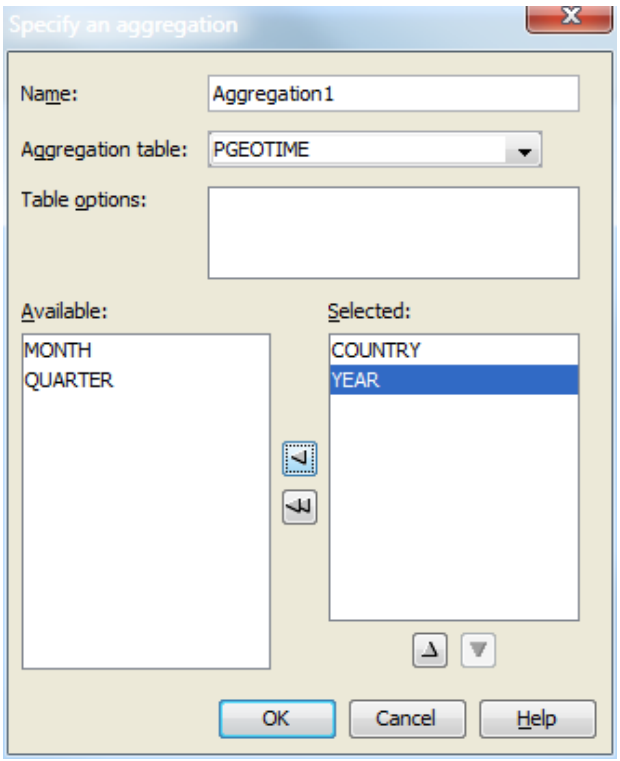


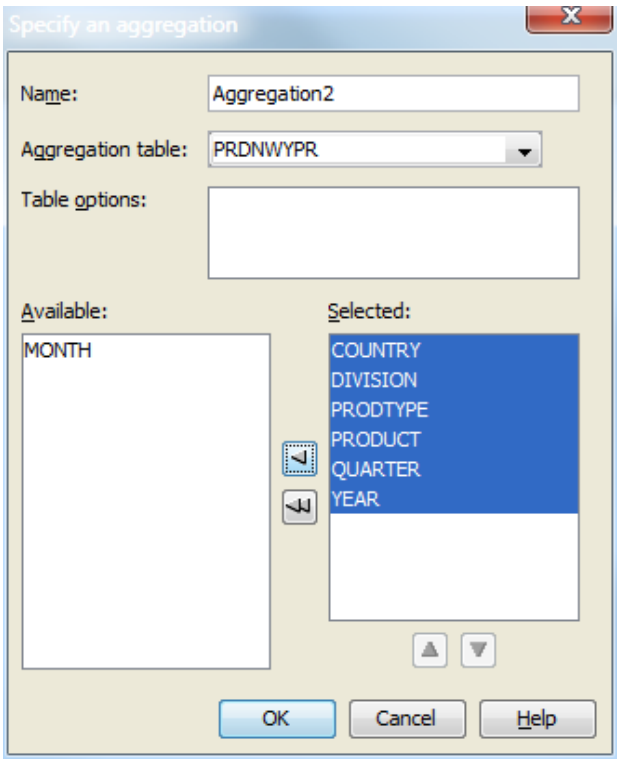
Define Stored Aggregations

You can now define stored aggregations for the cube. Stored aggregations are aggregations that are stored in the aggregation tables. On the Cube Designer - Stored Aggregations page, click **Add** to create a stored aggregation.

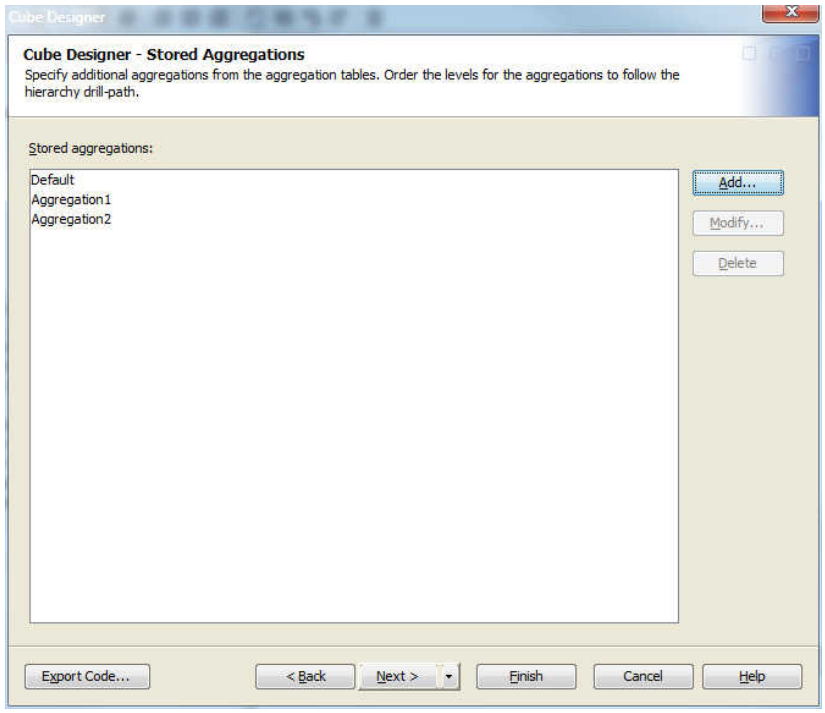


In the **Specify an aggregation** dialog box, enter the aggregation name, aggregation table, and any necessary table options for the aggregation. Then select the levels that are used in the aggregation. For this example, the stored aggregations named Aggregation1 and Aggregation2 are created.





The aggregations are listed on the Cube Designer – Stored Aggregations page.

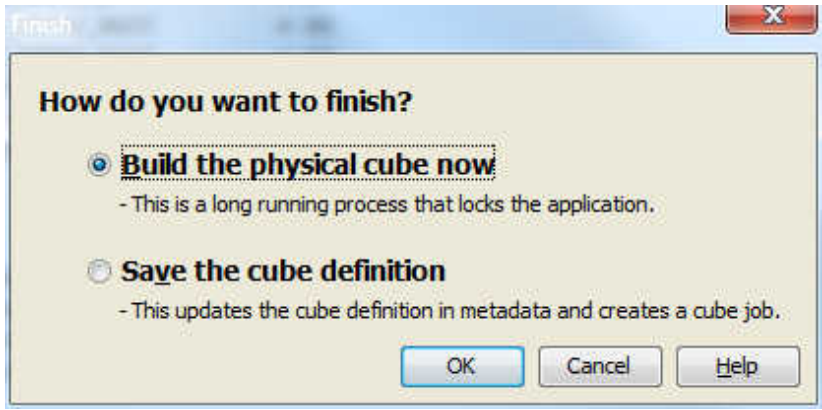


Define Aggregations

If needed, you can now define aggregations that are stored with the cube. Define the aggregations for the cube from the Cube Designer – Aggregations page.

Build the Cube

After you have finished entering information for the summarized cube, you can build the cube. On the Finish page, you can review the details of the cube that you just defined, and then save your PROC OLAP code to a file and click **Finish**. The Finish dialog box is displayed as shown.



Select one of the cube creation options and click **OK** to save and build the cube.

PROC OLAP CODE for the Summary Table Example

```
OPTIONS VALIDVARNAME=ANY;
LIBNAME olapsio BASE  "\\olap\tmp\libolap" ;

PROC OLAP
   CUBE                   = "/Shared Data/OLAPSchemas/SumCube"
   PATH                   = 'C:\v9cubes'
   DESCRIPTION            = 'Fully Summarized Cube'
   NONUPDATEABLE
   MAXTHREADS             = 5000
;


   METASVR
      HOST        = "J12345.na.sas.com"
      PORT        = 8561
      OLAP_SCHEMA = "SASApp - OLAP Schema";

   DIMENSION Geography
      CAPTION          = 'Geography'
      SORT_ORDER       = ASCENDING
      HIERARCHIES      = (
         Geography
         ) /* HIERARCHIES */;

      HIERARCHY Geography
         ALL_MEMBER = 'All Geography'
         CAPTION    = 'Geography'
         LEVELS     = (
            COUNTRY DIVISION
            ) /* LEVELS */
         DEFAULT;

      LEVEL COUNTRY
         FORMAT         =  $CHAR10.
         CAPTION        =  'Country'
         SORT_ORDER     =  ASCENDING;

      LEVEL DIVISION
         FORMAT         =  $CHAR10.
         CAPTION        =  'Division'
         SORT_ORDER     =  ASCENDING;

   DIMENSION Products
      CAPTION          = 'Products'
      SORT_ORDER       = ASCENDING
      HIERARCHIES      = (
         Products
```

```
      ) /* HIERARCHIES */;

   HIERARCHY Products
      ALL_MEMBER = 'All Products'
      CAPTION    = 'Products'
      LEVELS     = (
         PRODUCT PRODTYPE
         ) /* LEVELS */
      DEFAULT;

   LEVEL PRODUCT
      FORMAT         =  $CHAR10.
      CAPTION        =  'Product'
      SORT_ORDER     =  ASCENDING;

   LEVEL PRODTYPE
      FORMAT         =  $CHAR10.
      CAPTION        =  'Product type'
      SORT_ORDER     =  ASCENDING;

DIMENSION Dates
   CAPTION          = 'Dates'
   TYPE             = TIME
   SORT_ORDER       = ASCENDING
   HIERARCHIES      = (
      Dates
      ) /* HIERARCHIES */;

   HIERARCHY Dates
      ALL_MEMBER = 'All Dates'
      CAPTION    = 'Dates'
      LEVELS     = (
         YEAR QUARTER MONTH
         ) /* LEVELS */
      DEFAULT;

   LEVEL YEAR
      FORMAT         =  4.
      TYPE           =  YEAR
      CAPTION        =  'Year'
      SORT_ORDER     =  ASCENDING;

   LEVEL QUARTER
      FORMAT         =  8.
      TYPE           =  QUARTERS
      CAPTION        =  'Quarter'
      SORT_ORDER     =  ASCENDING;

   LEVEL MONTH
      FORMAT         =  MONNAME3.
      TYPE           =  MONTHS
      CAPTION        =  'Month'
      SORT_ORDER     =  ASCENDING;

MEASURE predsumSUM
   STAT        = SUM
```

```
   ANALYSIS    = PredictGroup
   AGGR_COLUMN = predsum
   CAPTION     = 'Sum of predsum'
   FORMAT      = Best12.
   DEFAULT;

MEASURE prednN
   STAT        = N
   ANALYSIS    = PredictGroup
   AGGR_COLUMN = predn
   CAPTION     = 'Number of values for predn'
   FORMAT      = 12.;

MEASURE actsumSUM
   STAT        = SUM
   ANALYSIS    = ActualGroup
   AGGR_COLUMN = actsum
   CAPTION     = 'Sum of actsum'
   FORMAT      = Best12.;

MEASURE actnN
   STAT        = N
   ANALYSIS    = ActualGroup
   AGGR_COLUMN = actn
   CAPTION     = 'Number of values for actn'
   FORMAT      = 12.;

MEASURE PredictedDerived
   STAT        = AVG
   ANALYSIS    = PredictGroup
   CAPTION     = 'Average PredictGroup'
   FORMAT      = Best12.;

MEASURE ActualDerived
   STAT        = AVG
   ANALYSIS    = ActualGroup
   CAPTION     = 'Average ActualGroup'
   FORMAT      = Best12.;

AGGREGATION /* Default */
   /* levels */
   COUNTRY DIVISION MONTH PRODTYPE
   PRODUCT QUARTER YEAR
   / /* options */
   TABLE     = olapsio.PRDNWYPR
   NAME      = 'Default';

AGGREGATION /* Aggregation1 */
   /* levels */
   COUNTRY YEAR
   / /* options */
   TABLE     = olapsio.PGEOTIME
   NAME      = 'Aggregation1';

AGGREGATION /* Aggregation2 */
   /* levels */
```

```
    COUNTRY DIVISION PRODTYPE
    PRODUCT QUARTER YEAR
    / /* options */
    TABLE     = olapsio.PRDNWYPR
    NAME      = 'Aggregation2';

RUN;
```

PROC OLAP Statements and Options for a Summary Table

The following table lists the PROC OLAP statements and options that you use to load cubes from a fully summarized data source. (A fully summarized data source contains a crossing of all dimensions. It is also known as an NWAY). Unlike a detail table or star schema, a fully summarized cube does not use either the DATA= or FACT= option. Instead, the TABLE= option is used in the AGGREGATION statement.

Table 9.3 *Statements and Options Used to Load Cubes from Fully Summarized Data*

Statements	Options	Required or Optional
PROC OLAP	CUBE=	Required
	PATH=	Required
	DESC=	Optional
	NO_NWAY	Optional
	WORKPATH=	Optional
	DT_TABLE	Optional
METASVR	OLAP_SCHEMA=	Required
	REPOSITORY=	Optional
	HOST=	Optional
	PORT=	Optional
	PROTOCOL=	Optional
	USERID=	Optional
	PW=	Optional
DIMENSION	HIERARCHIES=	Required

Statements	Options	Required or Optional
	DESC=	Optional
	CAPTION=	Optional
	TYPE=TIME	Required only for TIME dimensions
	SORT_ORDER=	Optional
LEVEL		The LEVEL statement is optional unless you want to specify time periods for each level in a TIME dimension. If you specify a time period for one level, then you must specify a time period for all levels. To specify a time period, you use the TYPE= option.
HIERARCHY	LEVELS=	Required
	DESC=	Optional
	CAPTION=	Optional
MEASURE	STAT=	Required
	COLUMN | ANALYSIS=	Required
	AGGR_COLUMN=	Required
	DESC=	Optional
	CAPTION=	Optional
	UNITS=	Optional
	FORMAT=	Optional
	DEFAULT=	Optional
PROPERTY	prop-name	Required
	LEVEL=	Required
AGGREGATION	Names of the contiguous levels to be used to create the aggregation	Required (additional AGGREGATION statements without the TABLE= option can be used to create aggregations other than the automatically defined NWAY).

Statements	Options	Required or Optional
	TABLE=	Required

Tuning Aggregations for a Cube

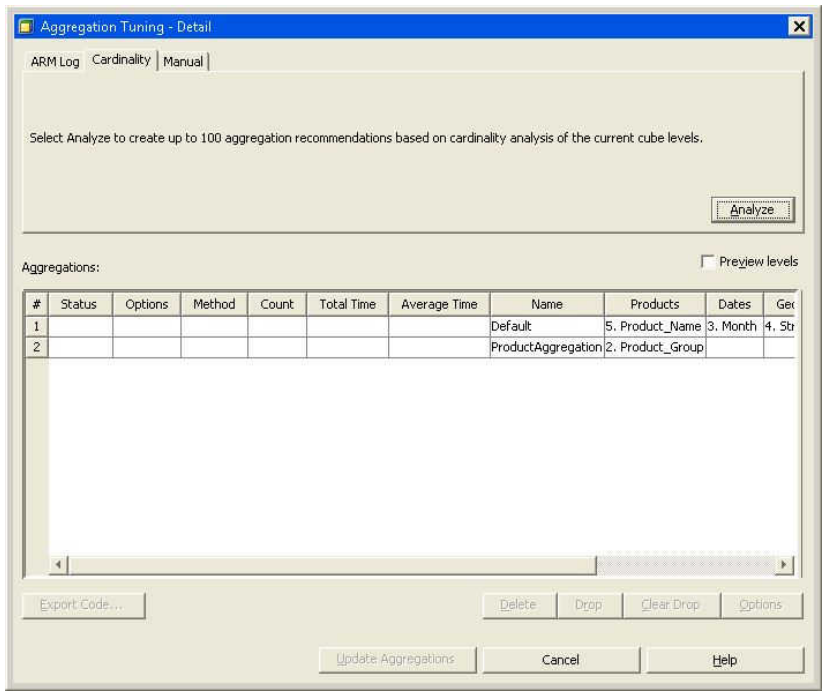
Overview

After you have built a cube, you can modify the aggregations for the cube with the Aggregation Tuning function in SAS OLAP Cube Studio. The Aggregation Tuning function enables you to either automatically generate aggregation recommendations or to manually define cube aggregations. In the SAS OLAP Cube Studio tree view, select a cube and select **Aggregation Tuning** from the **Actions** menu. In the Aggregation Tuning dialog box, you can select from three different methods of aggregation tuning:

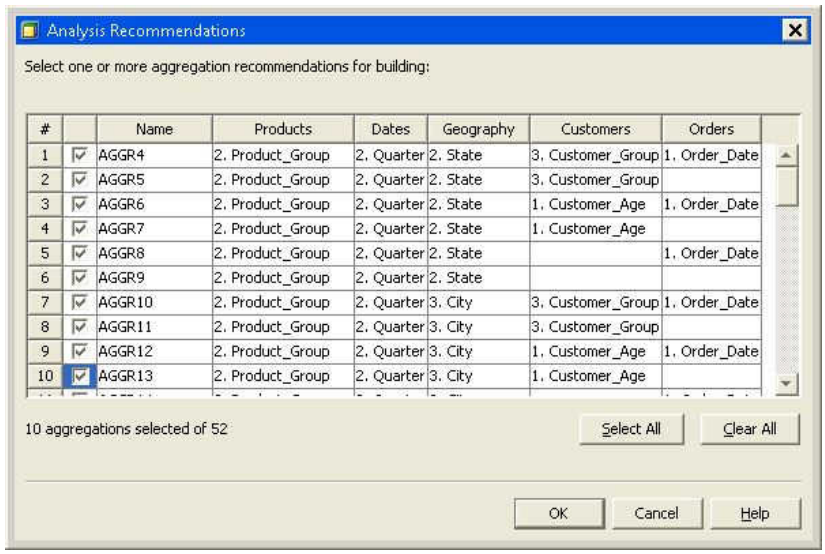
1. “Cardinality Tuning” on page 213
2. “Manual Tuning” on page 219
3. “Arm Log Tuning” on page 221

Cardinality Tuning

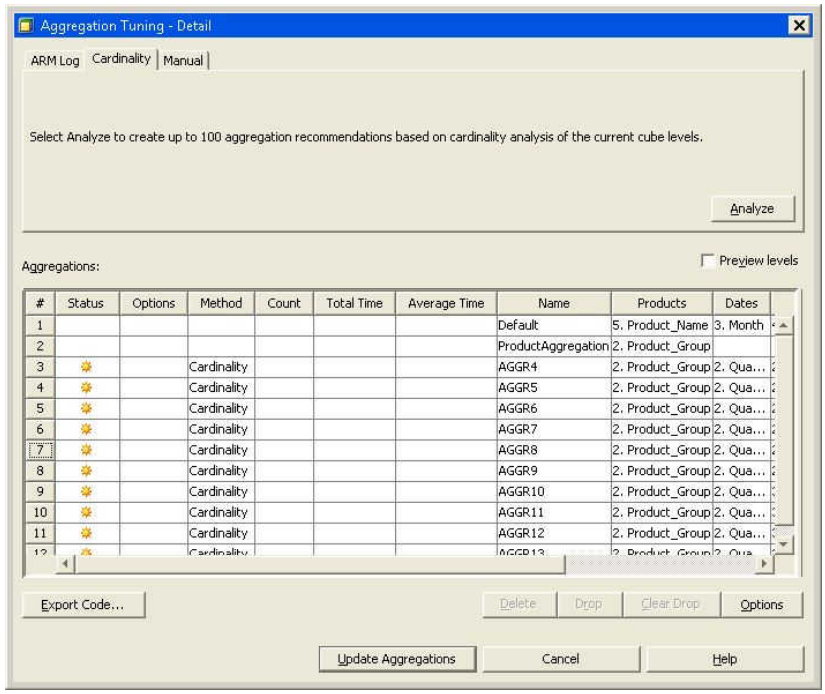
The **Cardinality** tab enables you to add aggregations that are recommended based on the relative cardinality (number of members) of the cube levels. This method of adding aggregations is used when a cube is first created and before an ARM log can be generated. On the **Cardinality** tab, click **Analyze**.



The Analysis Recommendations dialog box appears. A list of recommended aggregations is displayed. This list can contain up to 100 aggregation recommendations for the cube. Select the aggregation recommendations that you want to add to the Aggregations table. Click **Select All** to choose all of the aggregation recommendations. Select **OK** to execute the analysis. The selected aggregation recommendations are added to the list of aggregations in the **Aggregations** table.

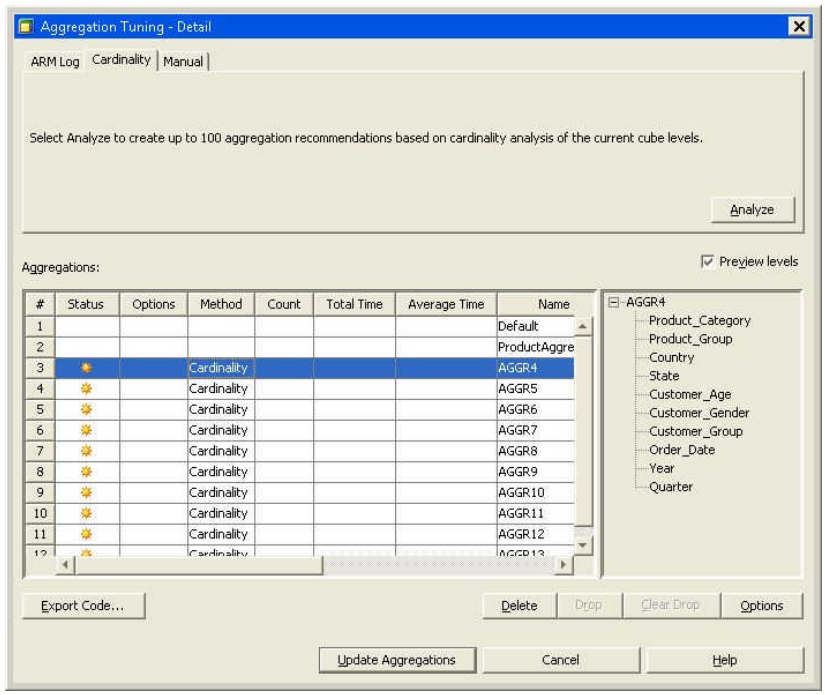


In the Aggregation Tuning dialog box, the **Aggregations** table displays the newly recommended aggregations. In the **Status** column, the new aggregation symbol is listed for each new aggregation. The **Method** column lists the recommended aggregations as **Cardinality** type aggregations. You can scroll through the list of aggregations to analyze which aggregations to keep, edit properties for aggregations, or remove aggregations.

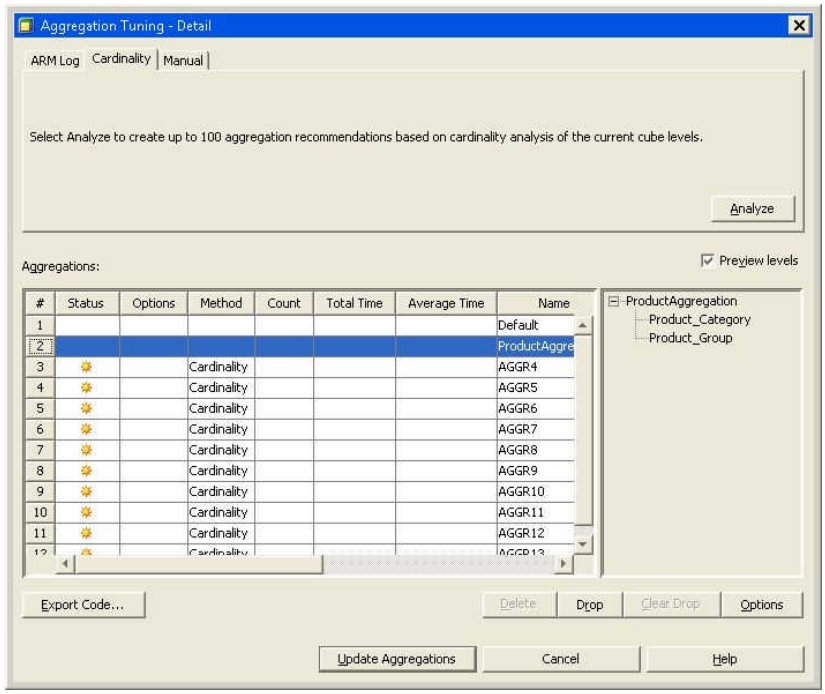


You can also view the levels that compose the aggregation recommendations. Select the **Preview levels** check box on the Aggregation Tuning dialog box. The **Preview levels** panel opens, displaying the levels for the currently selected aggregation recommendation.

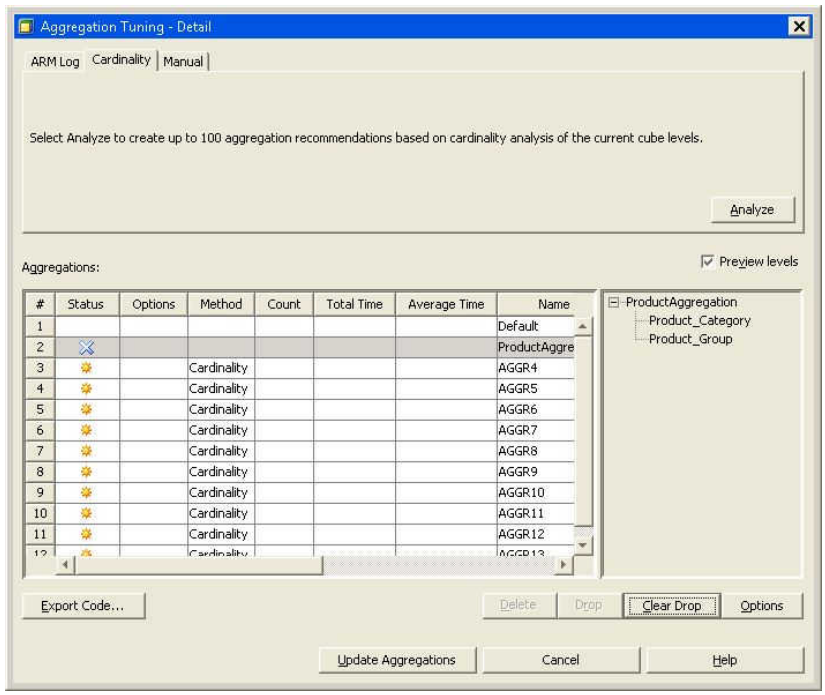
Note: The **Preview levels** panel is read-only.



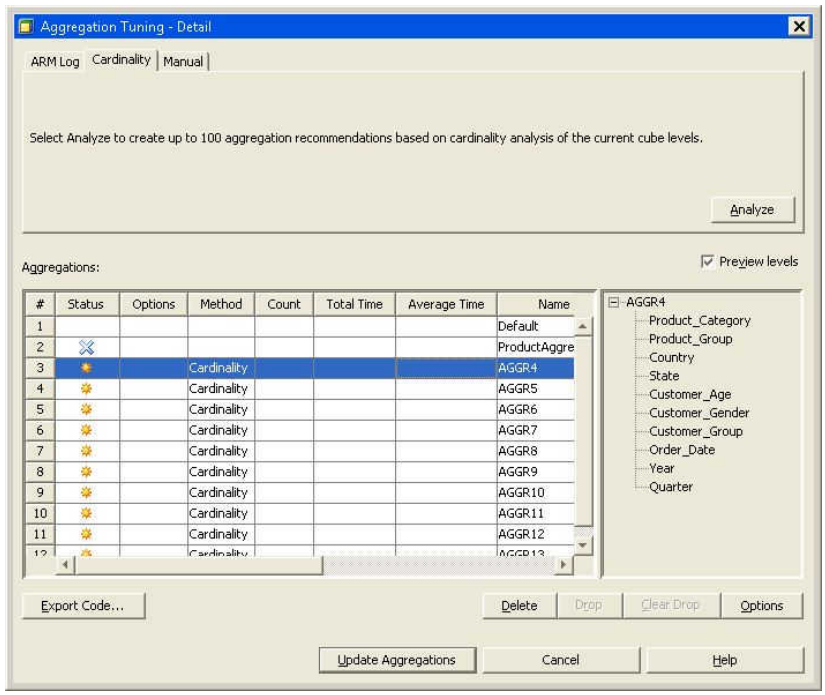
In the Aggregation Tuning dialog box, you can select to edit or drop aggregations that are loaded with the cube. When you select an existing aggregation, the **Drop** button becomes active.



If you click **Drop**, the drop icon appears in the **Status** column for that aggregation. In addition, the **Clear Drop** button becomes active. This button enables you to clear the drop status for an aggregation. Those aggregations that are marked with the drop icon are dropped from the cube the next time you click the **Build Aggregations** button.

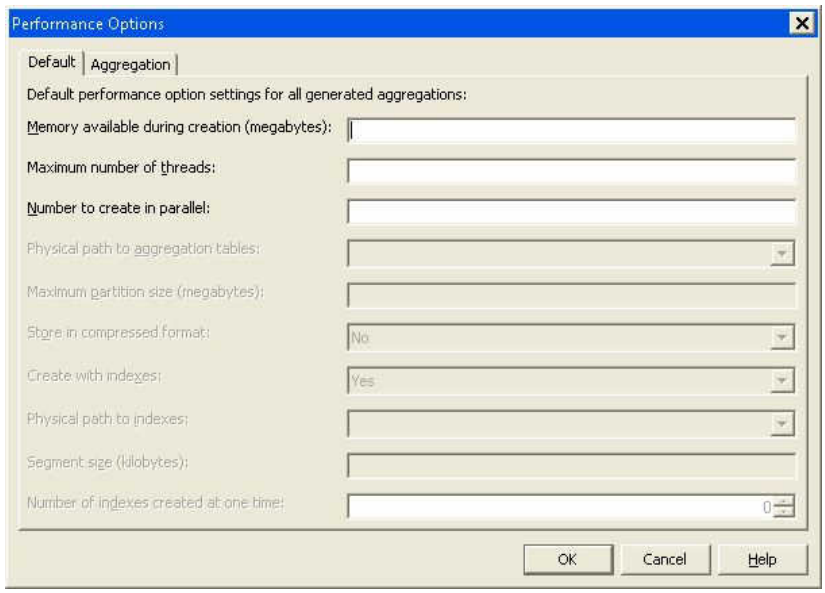


You can also review those aggregations that were recommended by cardinality analysis, and if needed delete them. When you select a newly recommended aggregation in the **Aggregations** table, the **Delete** button becomes active.

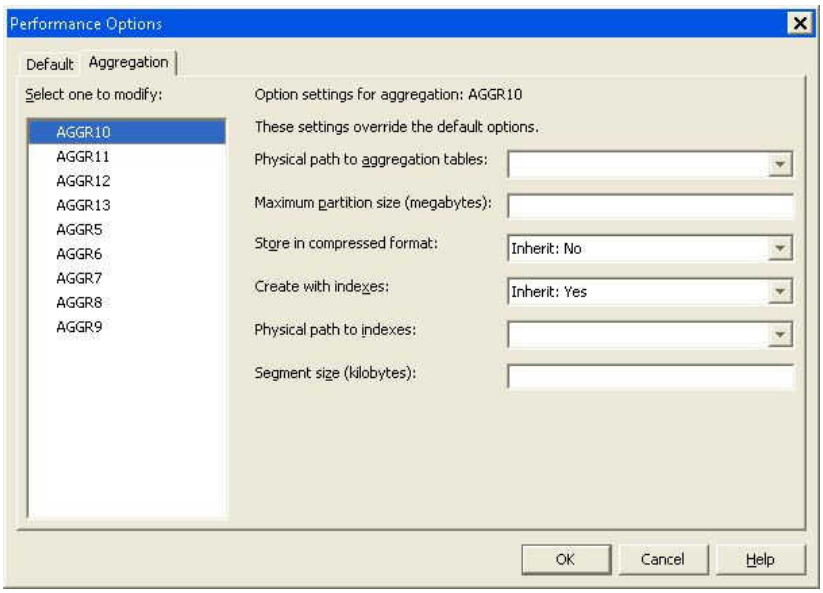


If you click **Delete**, a message dialog box appears. Select **Yes** to delete the selected aggregations. Select **No** to cancel the Delete function.

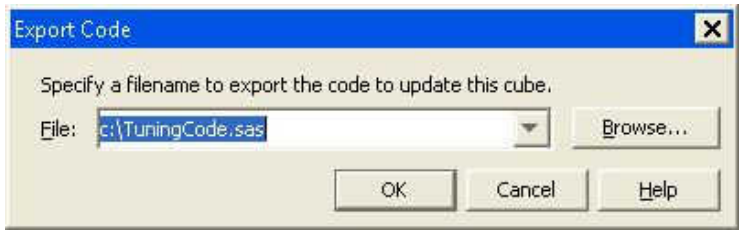
You can also modify the performance options for the cube aggregations. In the Aggregation Tuning dialog box, click **Options**. This opens the Performance Options dialog box. On the **Default** tab, you can set default performance option settings for all generated aggregations for the cube.



On the **Aggregation** tab, you can select an individual aggregation and change performance option settings for that specific aggregation. Select **OK** when you are finished changing the performance options. The Performance Options dialog box closes.



If needed, you can export the SAS code that is used to build the aggregations that are listed in the **Aggregations** table. Select **Export Code** on the Aggregation Tuning dialog box. The Export Code dialog box appears. Enter the path of the text file that you are exporting the code to. Click **OK** to create the file. The code is stored in a text file that you can further review, edit, and use to recreate the cube later.

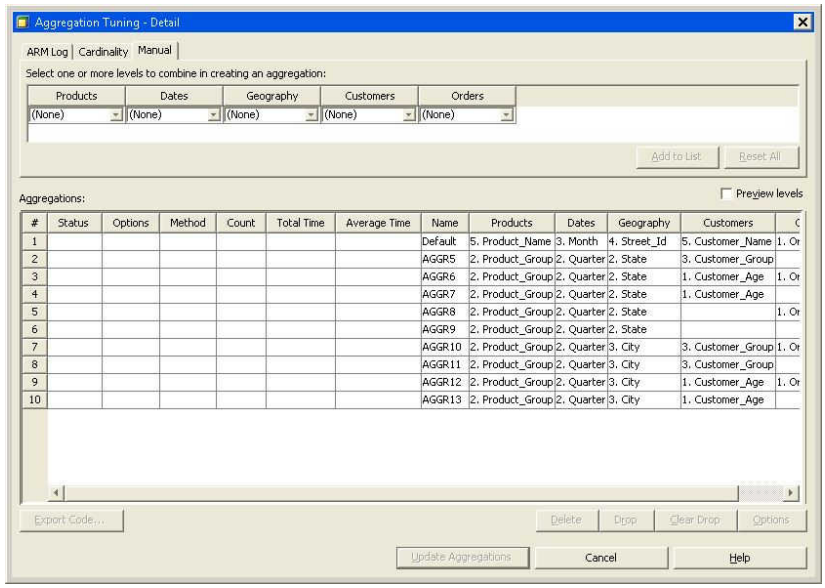


When you have finished updating and modifying the aggregations listed in the **Aggregations** table, you can click **Build Aggregations**. The Aggregation Tuning function builds the recommended aggregations, and drops any existing aggregations that are marked with the drop icon. A build success confirmation message opens and the Aggregation Tuning dialog box closes. Click **OK** on the message box.

Manual Tuning

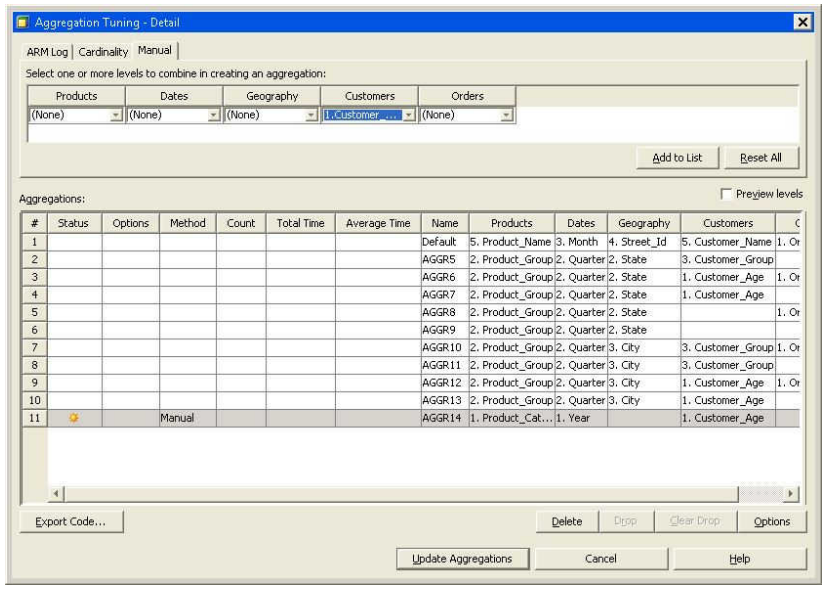
The **Manual** tab enables you to select the exact hierarchies and levels that you want use to generate aggregation recommendations from. The manual tuning option is used when you have dimension levels that are frequently used with other dimension levels. It is also used when a level has unique member counts (NUNIQUE) set.

Select the **Manual** tab on the Aggregation Tuning dialog box. The hierarchies for the cube are listed individually as columns. Levels for the hierarchies are numerically listed in drop-down lists on the columns. When you select an individual level from a hierarchy, you are selecting that level and its parent levels.

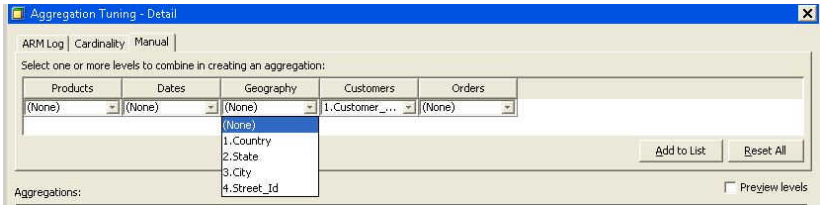


#	Status	Options	Method	Count	Total Time	Average Time	Name	Products	Dates	Geography	Customers	
1							Default	5. Product_Name	3. Month	4. Street_Id	5. Customer_Name	1. Or
2							AGGR5	2. Product_Group	2. Quarter	2. State	3. Customer_Group	
3							AGGR6	2. Product_Group	2. Quarter	2. State	1. Customer_Age	1. Or
4							AGGR7	2. Product_Group	2. Quarter	2. State	1. Customer_Age	
5							AGGR8	2. Product_Group	2. Quarter	2. State		1. Or
6							AGGR9	2. Product_Group	2. Quarter	2. State		
7							AGGR10	2. Product_Group	2. Quarter	3. City	3. Customer_Group	1. Or
8							AGGR11	2. Product_Group	2. Quarter	3. City	3. Customer_Group	
9							AGGR12	2. Product_Group	2. Quarter	3. City	1. Customer_Age	1. Or
10							AGGR13	2. Product_Group	2. Quarter	3. City	1. Customer_Age	

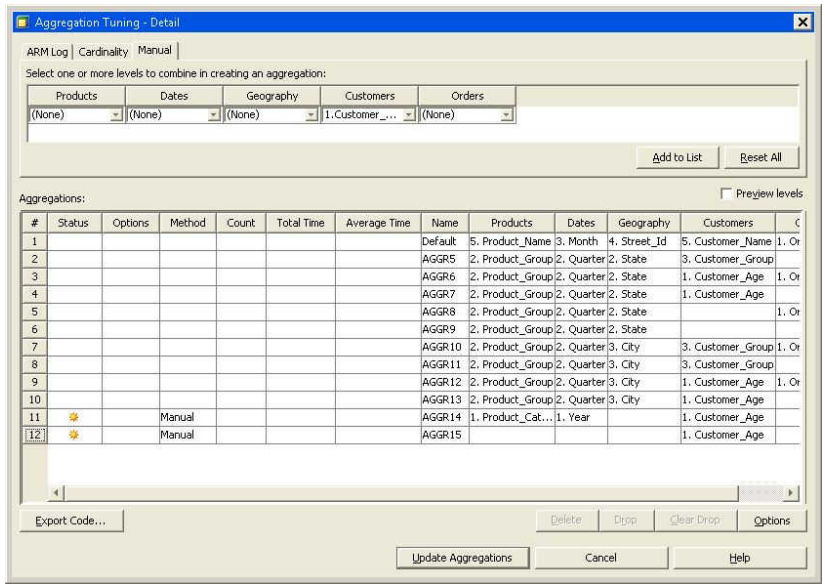
From the drop-down lists, select the levels that you want to use to create the aggregation recommendation. Both the **Add to List** and the **Reset All** buttons are enabled whenever one or more of the drop-down lists are changed to a selection other than the default selection of **(None)**.



For this example, select the levels for **Product_Category**, **Year**. Select **Add to List**.



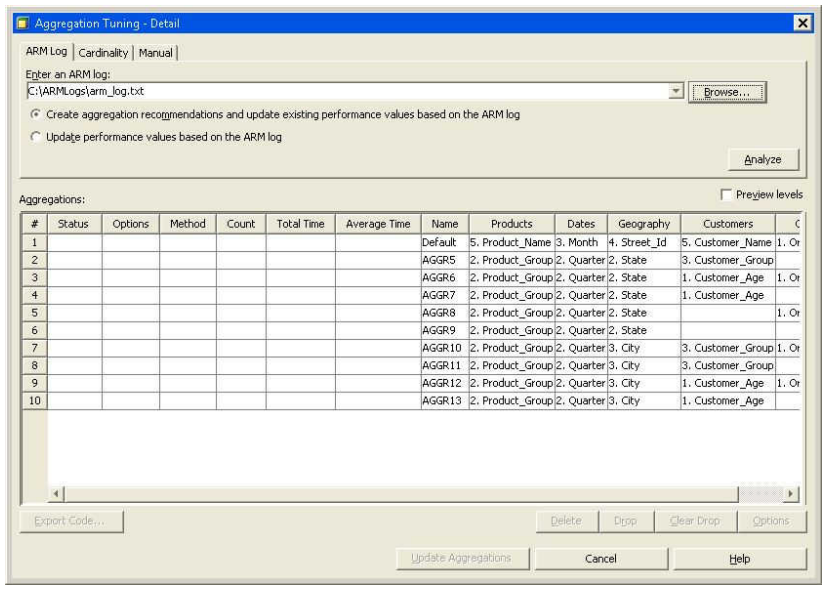
The aggregation recommendation that you created is listed in the **Aggregations** table. On the row for that aggregation, the levels that were manually selected for the aggregation are displayed and the new aggregation icon is listed in the **Status** column. When you are finished adding aggregation recommendations, click **Build Aggregation** to build the aggregations for the cube.



Arm Log Tuning

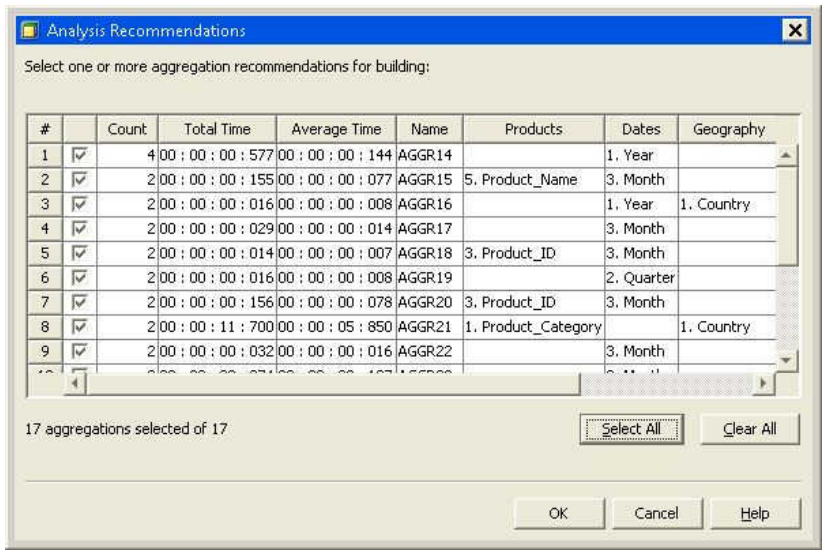
The **ARM Log** tab enables you to add aggregations that are based on the query analysis records that are stored in an Application Response Measuring (ARM) log. ARM analysis is recommended for optimal cube tuning. When the ARM API is used, a log file can be created that can be gleaned for query performance details. The ARM log is used to analyze query patterns and determine which aggregations to generate that will most likely have a positive impact on query performance. In order to create an ARM log, ARM logging must be turned on and queries must be performed against the cube. This function is only as effective as the amount of ARM data that is provided. Effectiveness also depends on whether the ARM log data truly reflects the future cube query patterns. For more information about creating the ARM log see “Using ARM to Monitor SAS OLAP Server Performance” in the *SAS 9.4 Intelligence Platform: System Administration Guide*.

Select the **ARM Log** tab and select the option **Create aggregation recommendations and update existing performance values based on the ARM log**. In the **Enter an ARM log** text field, enter the file path for the ARM log that you will use to generate aggregation recommendations. Use the **Browse** button to search for an ARM log if needed. After you have selected an ARM log file, click **Analyze**.



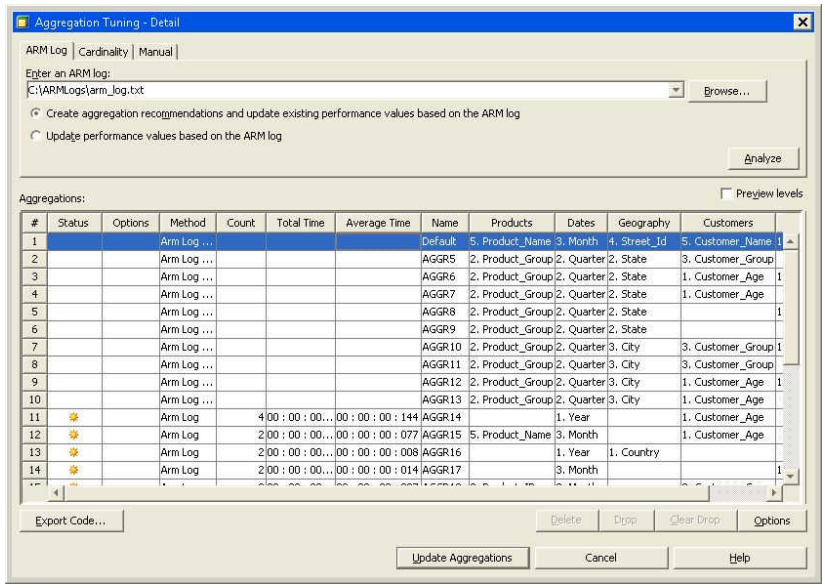
#	Status	Options	Method	Count	Total Time	Average Time	Name	Products	Dates	Geography	Customers	
1							Default	5. Product_Name	3. Month	4. Street_Id	5. Customer_Name	1. Or
2							AGGR5	2. Product_Group	2. Quarter	2. State	3. Customer_Group	
3							AGGR6	2. Product_Group	2. Quarter	2. State	1. Customer_Age	1. Or
4							AGGR7	2. Product_Group	2. Quarter	2. State	1. Customer_Age	
5							AGGR8	2. Product_Group	2. Quarter	2. State		1. Or
6							AGGR9	2. Product_Group	2. Quarter	2. State		
7							AGGR10	2. Product_Group	2. Quarter	3. City	3. Customer_Group	1. Or
8							AGGR11	2. Product_Group	2. Quarter	3. City	3. Customer_Group	
9							AGGR12	2. Product_Group	2. Quarter	3. City	1. Customer_Age	1. Or
10							AGGR13	2. Product_Group	2. Quarter	3. City	1. Customer_Age	

After you have selected **Analyze**, the Analysis Recommendations dialog box appears. A list of recommended aggregations is displayed. Select the aggregation recommendations that you want to add to the **Aggregations** table. Click **Select All** to choose all of the aggregation recommendations. Click **OK** to execute the analysis. The selected aggregation recommendations are added to the list of aggregations in the **Aggregations** table.



#		Count	Total Time	Average Time	Name	Products	Dates	Geography
1	☑	4	00 : 00 : 00 : 577	00 : 00 : 00 : 144	AGGR14		1. Year	
2	☑	2	00 : 00 : 00 : 155	00 : 00 : 00 : 077	AGGR15	5. Product_Name	3. Month	
3	☑	2	00 : 00 : 00 : 016	00 : 00 : 00 : 008	AGGR16		1. Year	1. Country
4	☑	2	00 : 00 : 00 : 029	00 : 00 : 00 : 014	AGGR17		3. Month	
5	☑	2	00 : 00 : 00 : 014	00 : 00 : 00 : 007	AGGR18	3. Product_ID	3. Month	
6	☑	2	00 : 00 : 00 : 016	00 : 00 : 00 : 008	AGGR19		2. Quarter	
7	☑	2	00 : 00 : 00 : 156	00 : 00 : 00 : 078	AGGR20	3. Product_ID	3. Month	
8	☑	2	00 : 00 : 11 : 700	00 : 00 : 05 : 850	AGGR21	1. Product_Category		1. Country
9	☑	2	00 : 00 : 00 : 032	00 : 00 : 00 : 016	AGGR22		3. Month	

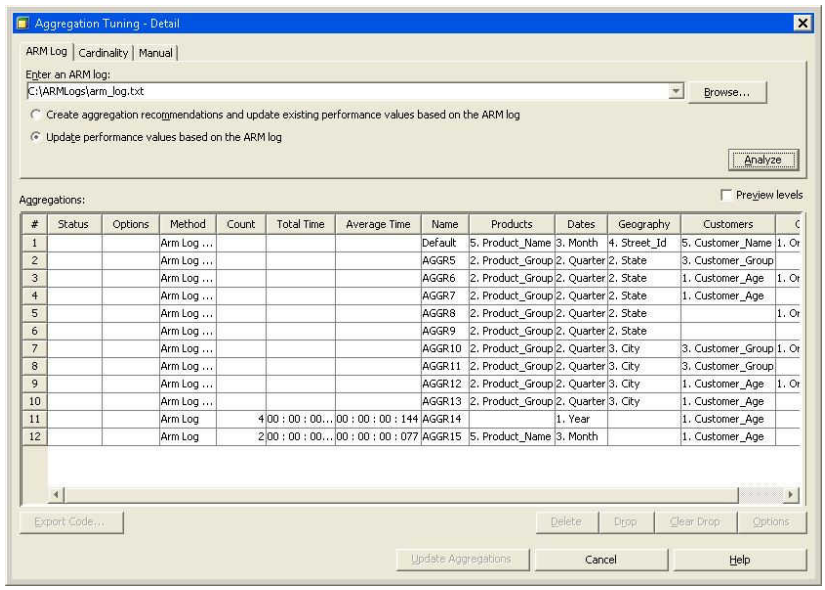
In the Aggregation Tuning dialog box, the aggregation recommendations are listed. The **Method** column displays the ARM log status for these aggregation recommendations. Select **Build Aggregations** to build the aggregations for the cube.



#	Status	Options	Method	Count	Total Time	Average Time	Name	Products	Dates	Geography	Customers
1			Arm Log ...				Default	5. Product_Name	3. Month	4. Street_Id	5. Customer_Name
2			Arm Log ...				AGGR5	2. Product_Group	2. Quarter	2. State	3. Customer_Group
3			Arm Log ...				AGGR6	2. Product_Group	2. Quarter	2. State	1. Customer_Age
4			Arm Log ...				AGGR7	2. Product_Group	2. Quarter	2. State	1. Customer_Age
5			Arm Log ...				AGGR8	2. Product_Group	2. Quarter	2. State	
6			Arm Log ...				AGGR9	2. Product_Group	2. Quarter	2. State	
7			Arm Log ...				AGGR10	2. Product_Group	2. Quarter	3. City	3. Customer_Group
8			Arm Log ...				AGGR11	2. Product_Group	2. Quarter	3. City	3. Customer_Group
9			Arm Log ...				AGGR12	2. Product_Group	2. Quarter	3. City	1. Customer_Age
10			Arm Log ...				AGGR13	2. Product_Group	2. Quarter	3. City	1. Customer_Age
11	✱		Arm Log	4	00 : 00 : 00...	00 : 00 : 00 : 144	AGGR14		1. Year		1. Customer_Age
12	✱		Arm Log	2	00 : 00 : 00...	00 : 00 : 00 : 077	AGGR15	5. Product_Name	3. Month		1. Customer_Age
13	✱		Arm Log	2	00 : 00 : 00...	00 : 00 : 00 : 008	AGGR16		1. Year	1. Country	
14	✱		Arm Log	2	00 : 00 : 00...	00 : 00 : 00 : 014	AGGR17		3. Month		

Alternately, you can also choose to update only the statistics for the existing aggregations that have entries in the ARM log. The **Update performance values based on the ARM log** option is used when you want to examine the information in the ARM log and verify that the existing aggregations are being used. This can help determine which aggregations to drop. Select **Update** to execute the update.

The existing aggregations that are listed in the **Aggregations** table have been updated in the **Count**, **Total Time**, and **Average Time** columns. The **Method** column also lists **Arm Log** as the update method.



Adding Data to a Cube with Cube Update

Overview

After you have built a cube with either a detail table or a star schema, you can update the cube with the Incremental Update function in SAS OLAP Cube Studio. You can choose to update the original cube or generate an updated generation (copy) of the cube. You can then coalesce (consolidate) the aggregation racks (SPD Engine tables) into a single SPD Engine table.

Update a Cube In-Place

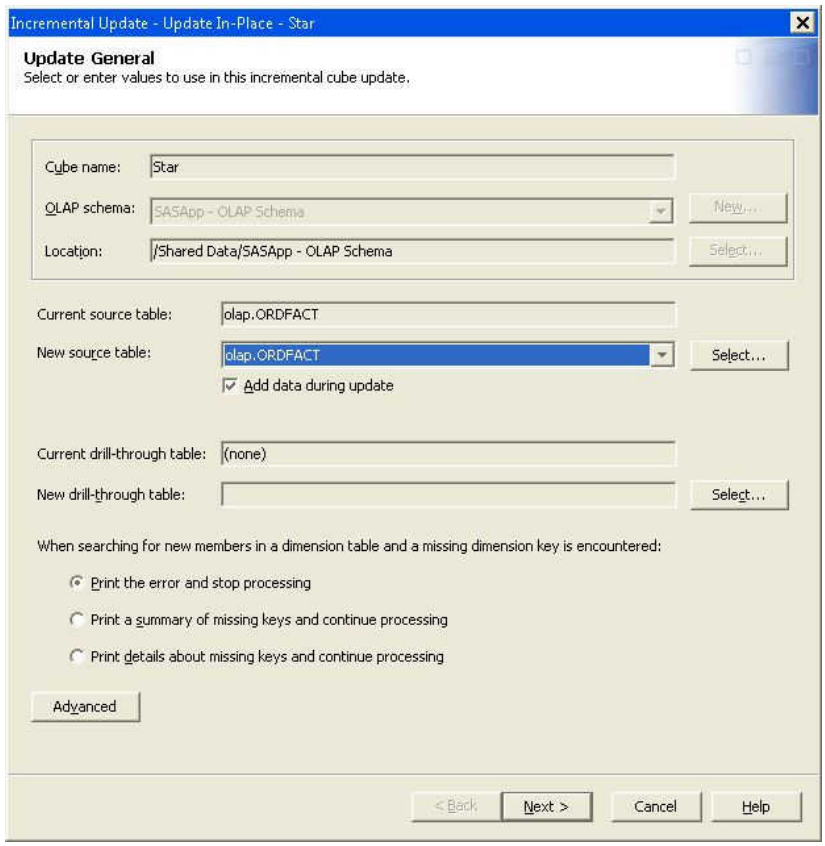
The Update In-Place function enables you to update the original cube. In SAS OLAP Cube Studio, select a cube from the tree view that you want to update. You can then right-click the cube and select **Incremental Update** ⇨ **Update In-Place**. You can also select the function from the **Actions** menu.

On the Update General page of the Incremental Update wizard, select the source table that you will use to update the cube with. If you are updating a star schema cube, the check box **Add data during update** is available. If selected, it activates the **New source table** box. Select a new source table. You can also click **Select** and choose a table that is not in the drop-down list.

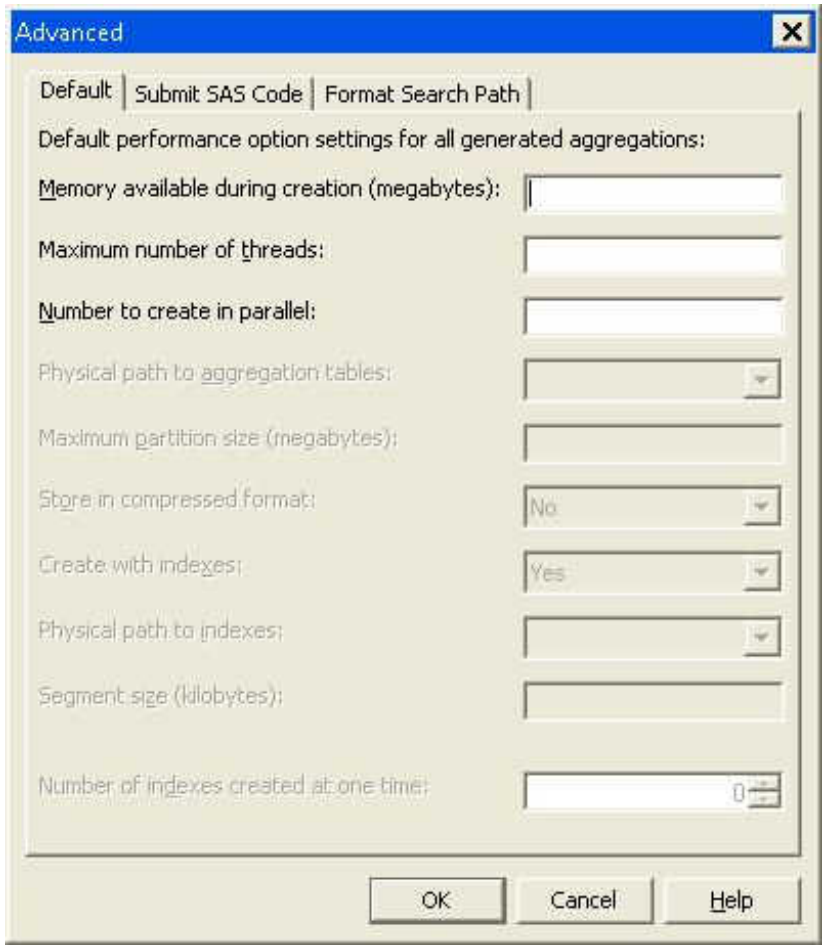
If a drill-through table is included in the cube update, click **Select** beside the **New drill-through table** text field and choose a drill-through table. If the cube was originally built without a drill-through table, the option `(none)` is listed in the **Current drill-through table** text field.

If you are updating a star schema cube, you can also specify how missing dimension keys are handled if encountered. Select one of the three radio button options under **When searching through new members in a table and a missing key is encountered**.

Note: Because you are updating the original cube with the Update In-Place function, the fields for **Cube Name**, **OLAP Schema**, and **Location** are inactive.

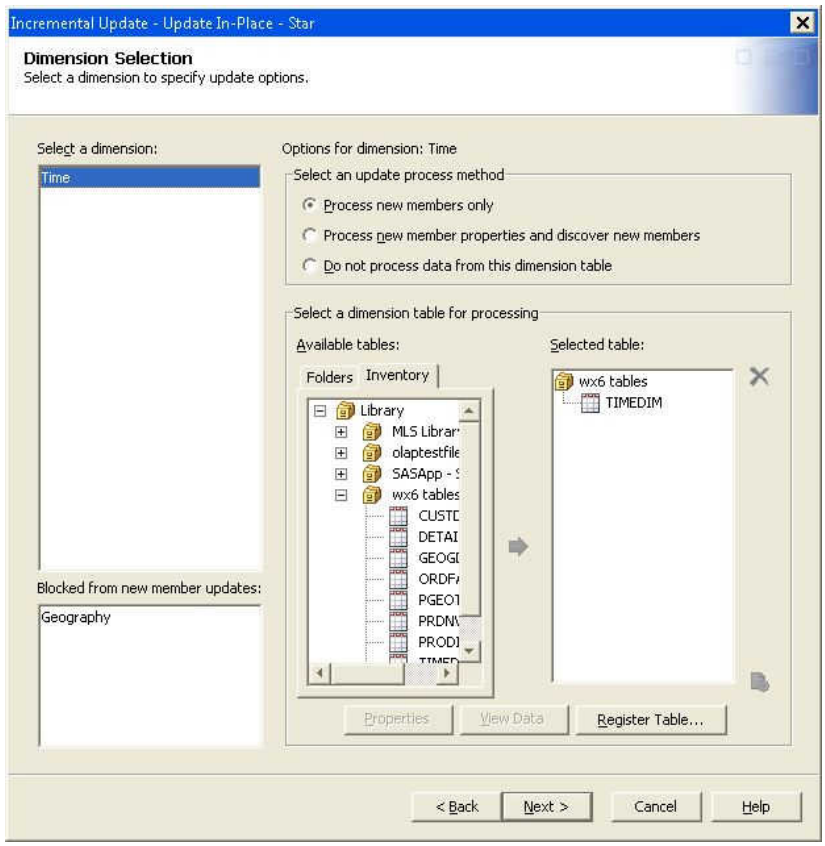


You can also click **Advanced** and specify aggregation performance options, any needed SAS code, and a search path for user-written formats to include with the cube update. Select **Next** when you are finished.

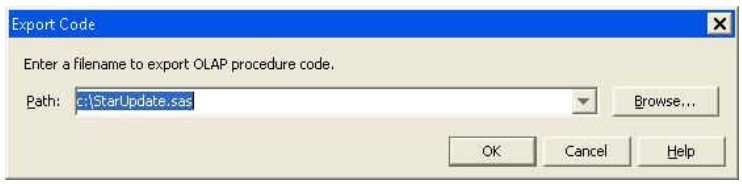


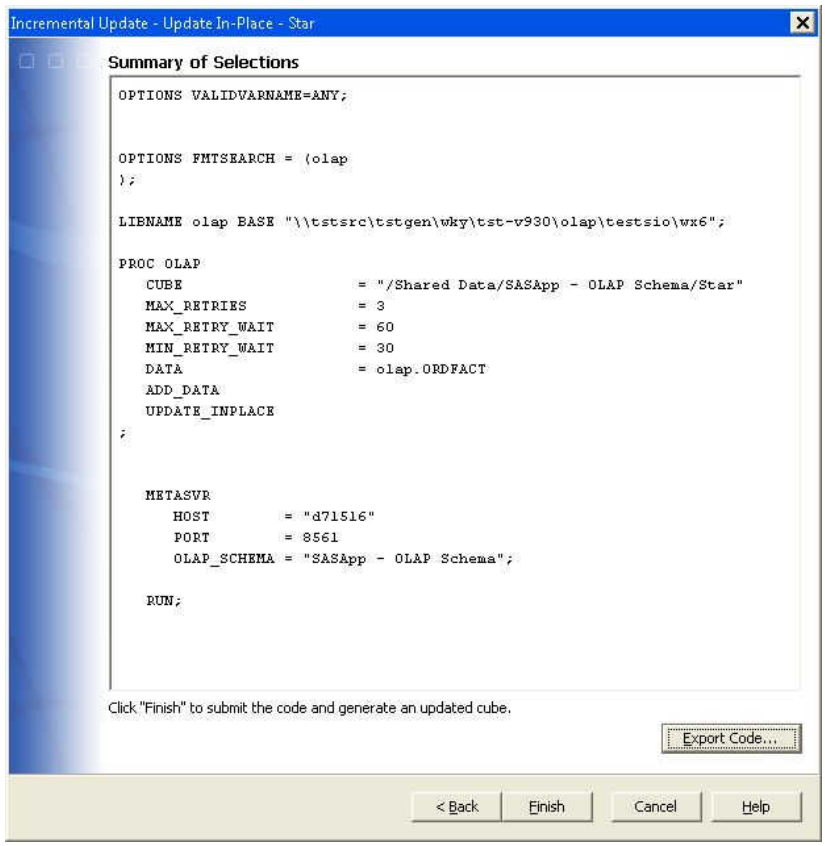
On the Dimension Selection page, you can click a dimension in the **Select a dimension** list. You can then select one of the radio buttons under **Select an update process method**. When the cube was originally built, you could also specify whether a dimension could be updated at a later time or not. Any dimensions that cannot be updated are listed in the **Blocked from new member updates** list and are not available for selection.

If you are updating a star schema cube, you can select a dimension table that can be used to update an individual dimension. Each dimension has a default table assigned from the original cube build. Using the left and right arrow keys, you can move tables between the **Available tables** and the **Selected table** lists. You can also view statistics about the table that you are selecting with the **Properties**, **View Data**, and **Table Options** buttons. If the table that you want to use is not included in the **Available Tables** list, you can register the table with the **Register Table** button. Select **Next** when you are finished.



On the Summary of Selections page, review the generated PROC OLAP code that is used to update the cube. You can save the PROC OLAP code to a file by selecting **Export Code**. Select the **Finish** to run the update of the cube.





Generating a New Cube

The Generate New Cube function enables you to create an updated copy of the original cube without modifying the original cube. It is useful for cubes that are in a production environment, as it enables you check the cube generation before users submit queries. In SAS OLAP Cube Studio, select a cube from the tree view that you want to update. You can then right-click the cube and select **Incremental Update** ⇨ **Generate New Cube**. You can also select the function from the **Actions** menu. This opens the Update General page of the Incremental Update wizard.

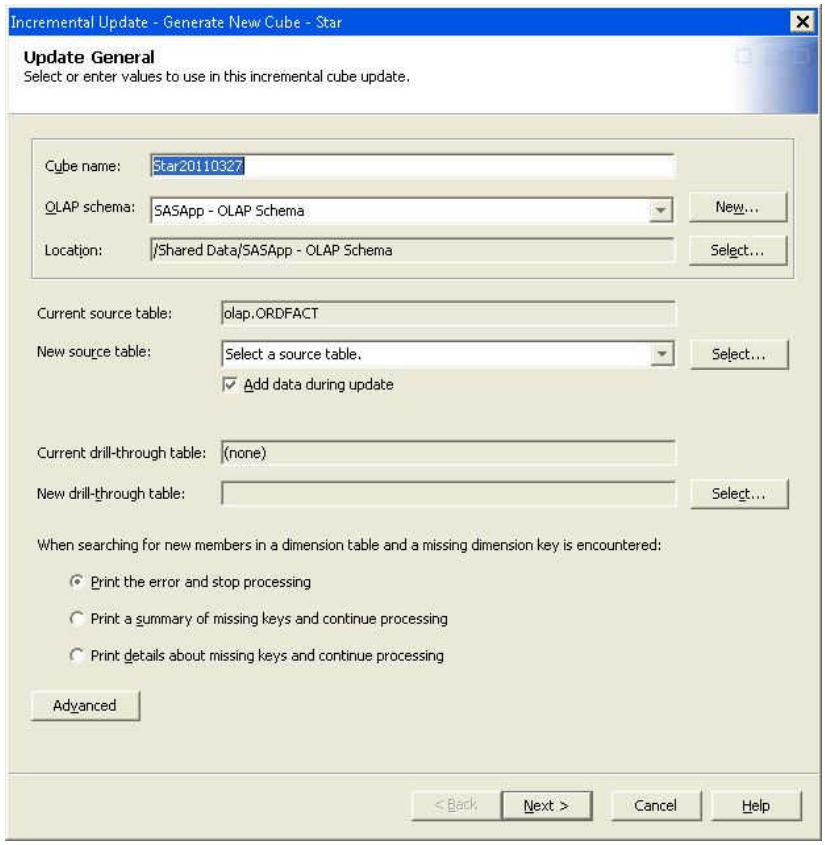
Because you are creating a generation of the original cube, you must enter a cube name that is different from the original cube in the **Cube Name** text box. You can also change the OLAP schema and the folder location for the new cube.

You can then select the source table that you will use to update the cube with. If you are updating a star schema cube, the check box **Add data during update** is available. If selected, it activates the **New source table** box. Select a new source table. You can also click **Select** and choose a table that is not in the drop-down list.

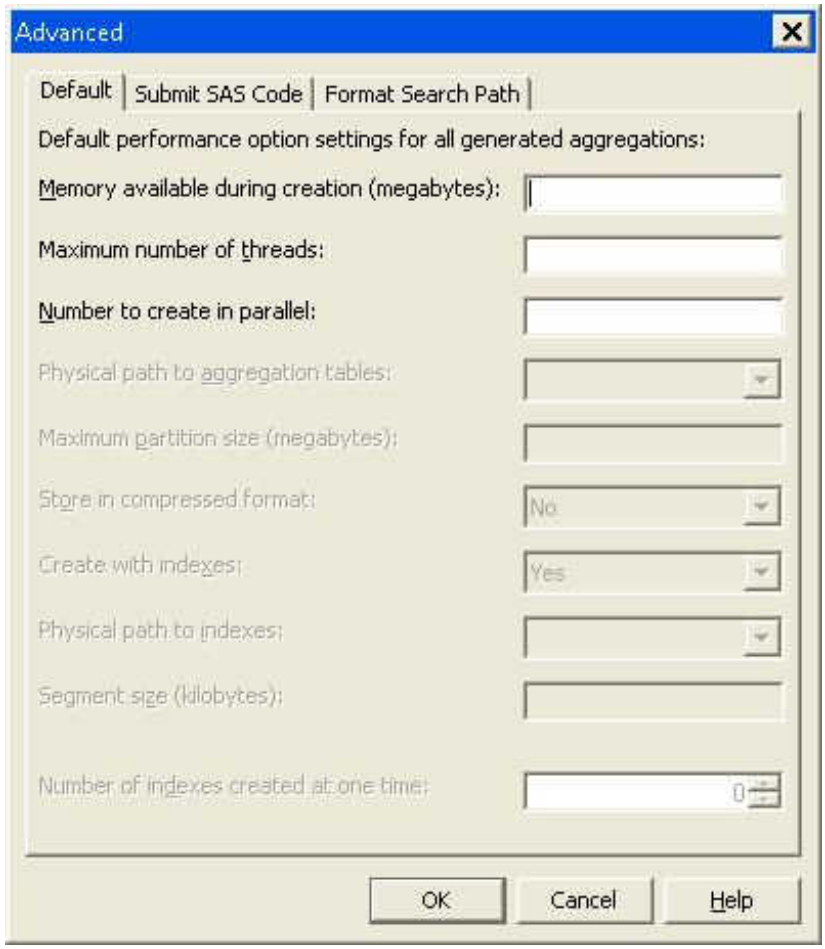
If a drill-through table is included in the cube update, click **Select** beside the **New drill-through table** text field and choose a drill-through table. If the cube was originally built without a drill-through table, the option `(none)` is listed in the **Current drill-through table** text field.

If you are updating a star schema cube, you can also specify how missing dimension keys are handled if encountered. Select one of the three radio button

options under **When searching through new members in a table and a missing key is encountered**.

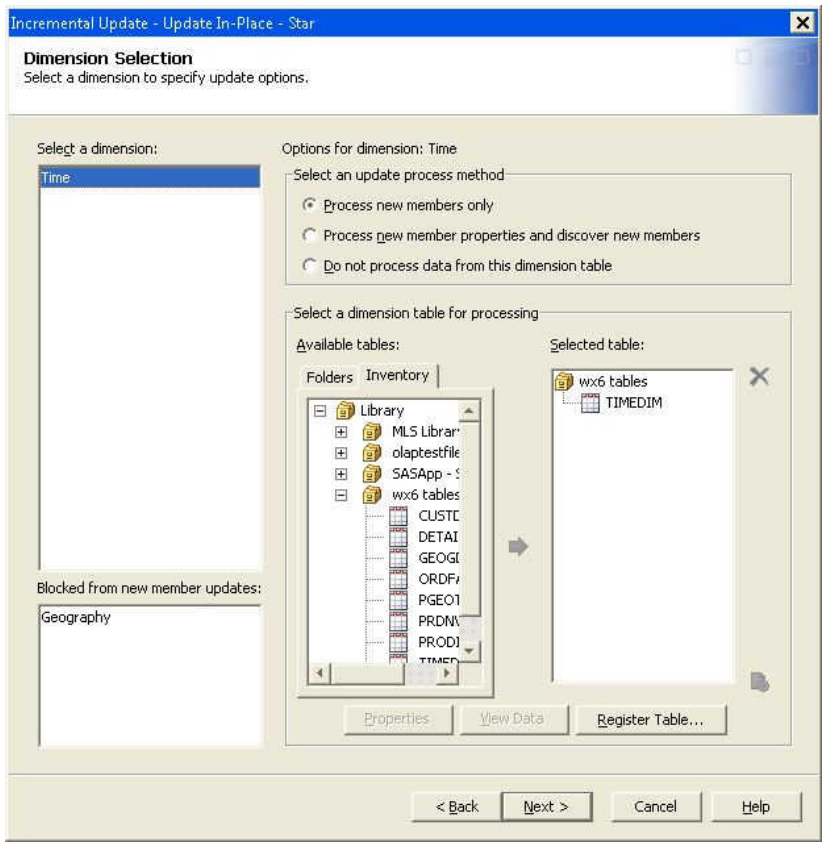


You can also click **Advanced** and specify aggregation performance options, any needed SAS code, and a search path for user-written formats to include with the cube update. Select **Next** when you are finished.

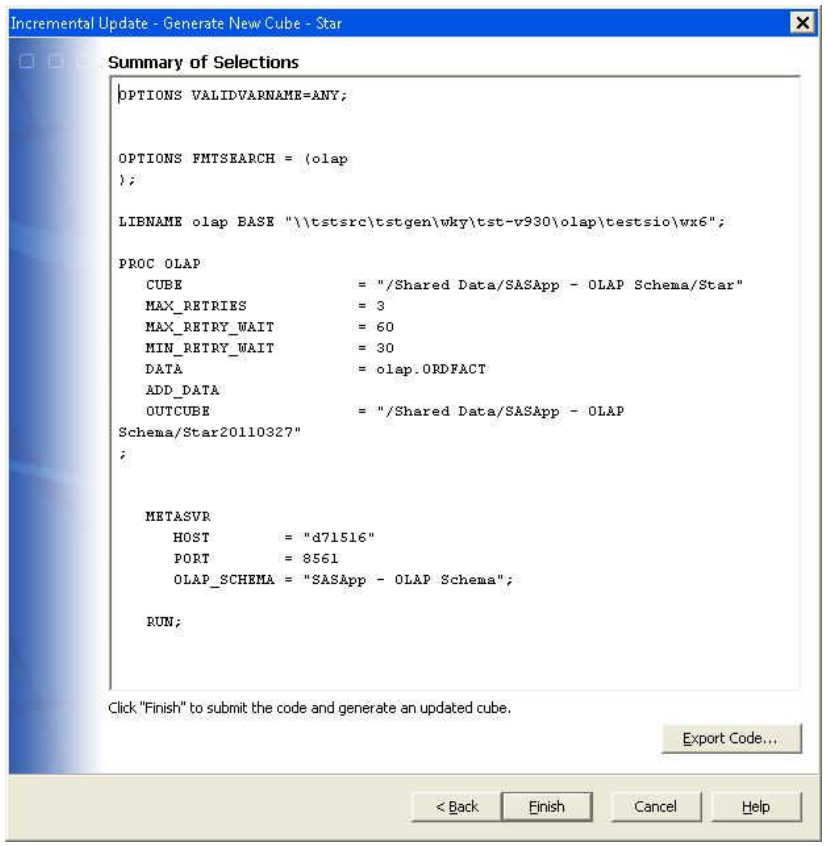


On the Dimension Selection page, you can click a dimension in the **Select a dimension** list. You can then select one of the radio button options under **Select an update process method**. When the cube was originally built, you could also specify whether a dimension could be updated at a later time or not. Any dimensions that cannot be updated are listed in the **Blocked from new member updates** list and are not available for selection.

If you are updating a star schema cube, you can select a dimension table that can be used to update an individual dimension. Each dimension has a default table assigned from the original cube build. Using the left and right arrow keys, you can move tables between the **Available tables** and the **Selected table** lists. You can also view statistics about the table that you are selecting with the **Properties**, **View Data**, and **Table Options** buttons. If the table that you want to use is not included in the **Available Tables** list, you can register the table with the **Register Table** button. Select **Next** when you are finished.



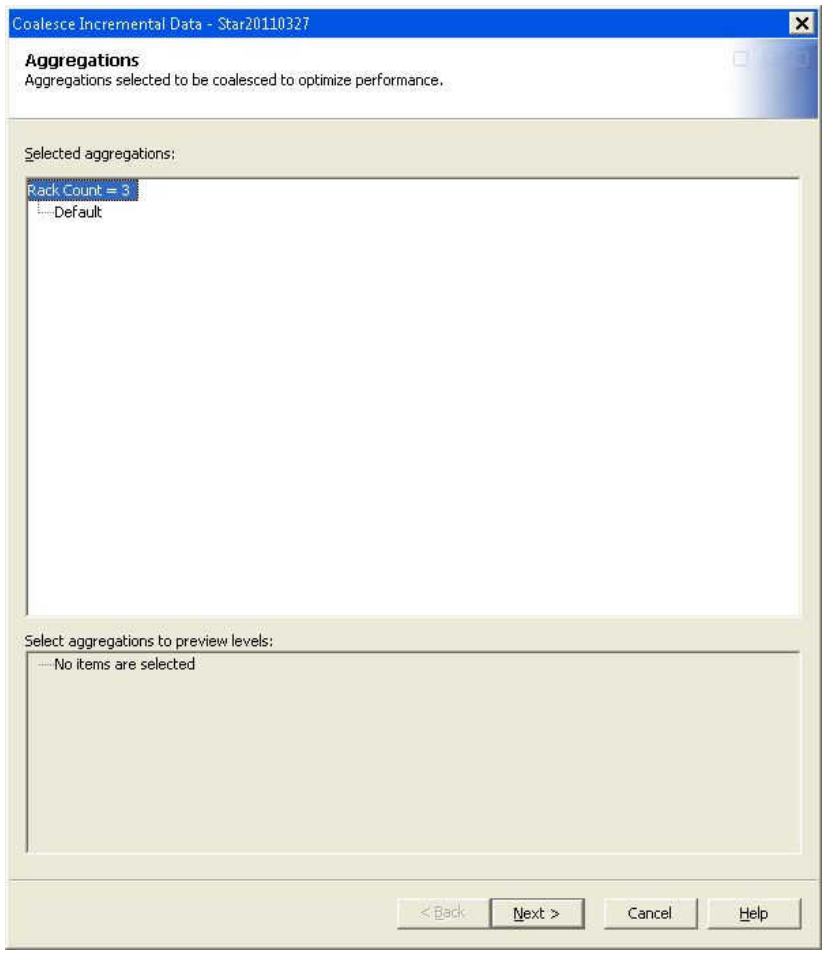
On the Summary of Selections page, review the generated PROC OLAP code that is used to update the cube. You can save the PROC OLAP code to a file by selecting **Export Code**. Select **Finish** to run the update of the cube.



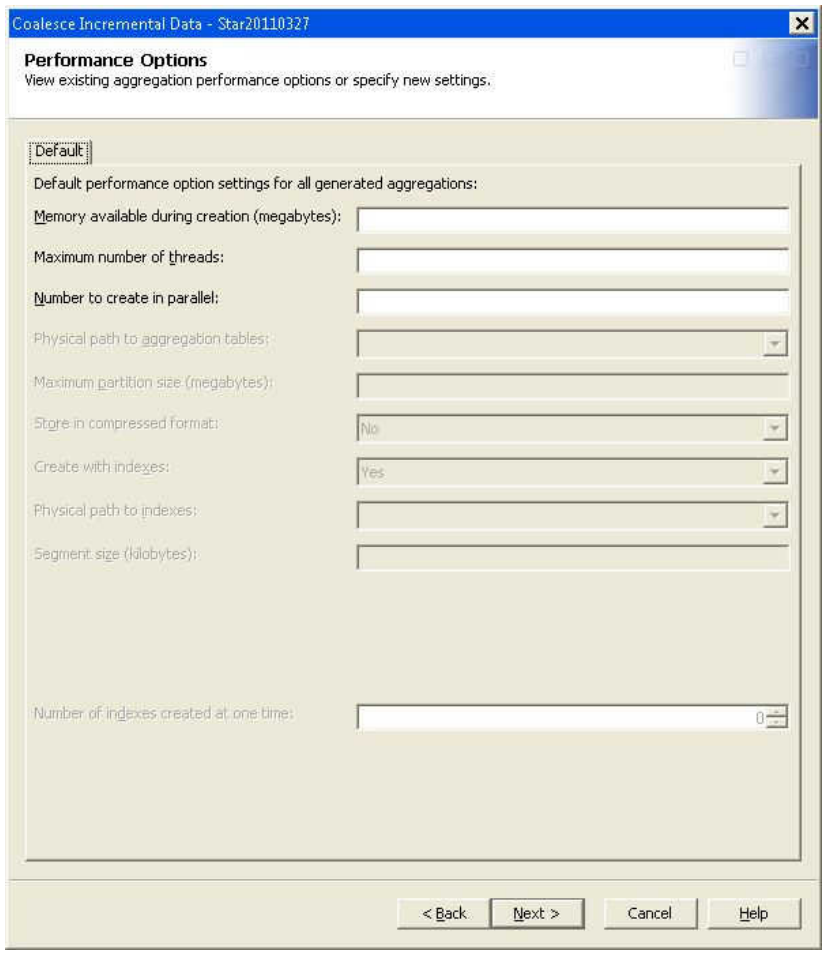
Coalesce Incremental Data for a Cube

After a cube has been updated, you can coalesce the aggregation racks (SPD Engine tables) into a single SPD Engine table. In SAS OLAP Cube Studio, select a cube from the tree view. You can then right-click the cube and select **Incremental Update** ⇨ **Coalesce Incremental Data**. You can also select the function from the **Actions** menu. This opens the Aggregations page of the Coalesce Incremental Data wizard.

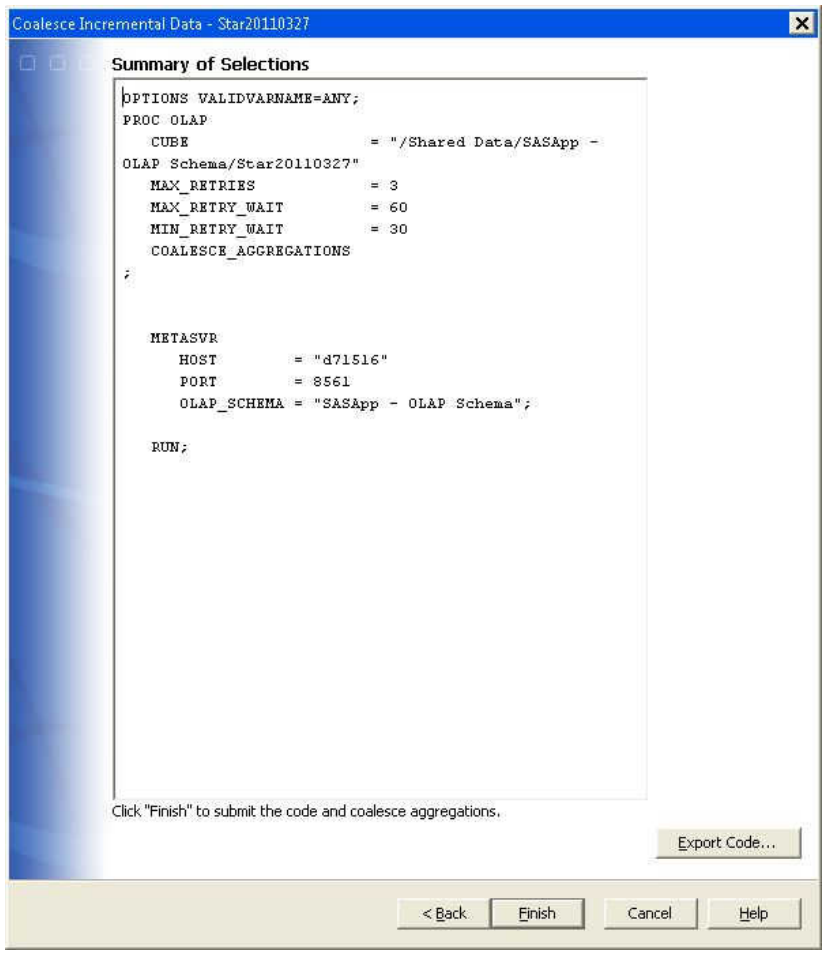
On the Aggregations page, you can view the aggregations that will be coalesced from the **Selected aggregations** list. You can also observe the levels of the selected aggregations in the **Select aggregations to preview levels** list. The preview list is display only. Select **Next**.



On the Performance Options page, you can specify query performance settings for the cube aggregations. The **Default** tab enables you to change performance settings for all aggregations of the cube. Enter any needed setting values. Select the **Next** when you are finished.



On the Summary of Selections page, review the generated PROC OLAP code that is used to coalesce the aggregations. You can save the PROC OLAP code to a file by selecting **Export Code**. Select **Finish** to run the update of the cube.



A progress dialog box displays after you click **Finish**.

The SAS log that is generated contains status information for the coalesce process.

```
31  NOTE: Aggregation "Default" was coalesced. Total records: 12734
32  NOTE: Cube "Star" was coalesced successfully.
33
34  NOTE: PROCEDURE OLAP used (Total process time):
35        real time           0.51 seconds
36        cpu time            0.68 seconds
37
38
```

Adding Calculated Members to a Cube

Overview

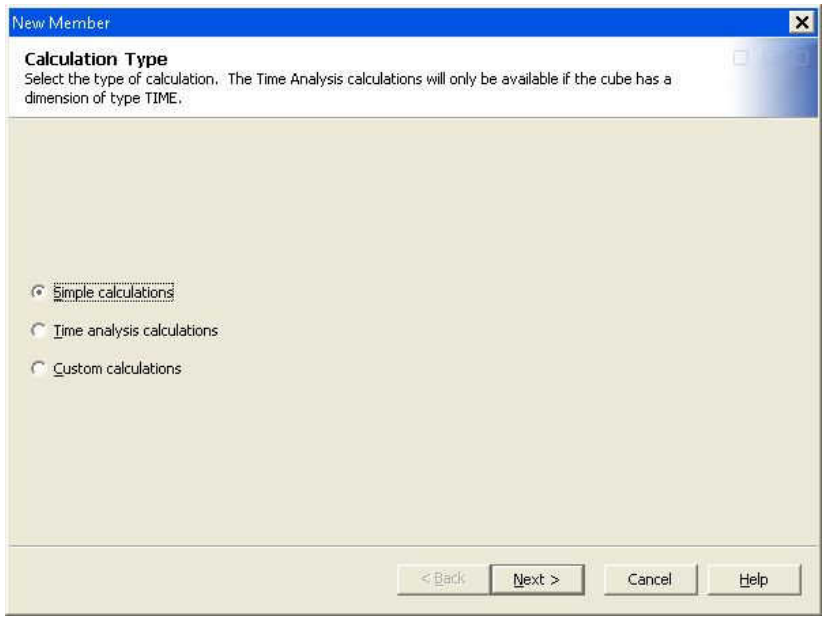
The Calculated Members function enables you to add new members for the Measures dimension. A calculated member is a definition (for a dimension member)

that you create and store with the cube. The calculated member value is generated at a later point, during query time. You can also define a calculated member as a measure. You can access the Calculated Members function from the SAS OLAP Cube Studio **Actions** menu or from the context menu for a cube.

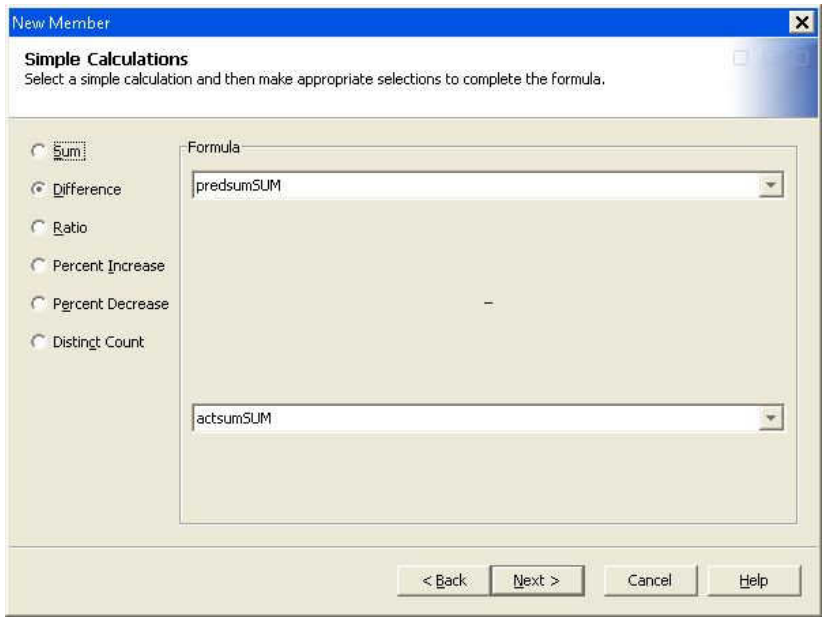
The following display shows the Calculations for Cube page of the Calculated Members function. The **Defined calculated members** list displays all calculated members that are defined in the metadata for the selected cube. On the Calculations for Cube page, you can add new calculated members or modify existing members for the selected cube. Select **Add** to launch the New Member wizard and define a new calculated member.

Creating a Simple Calculation

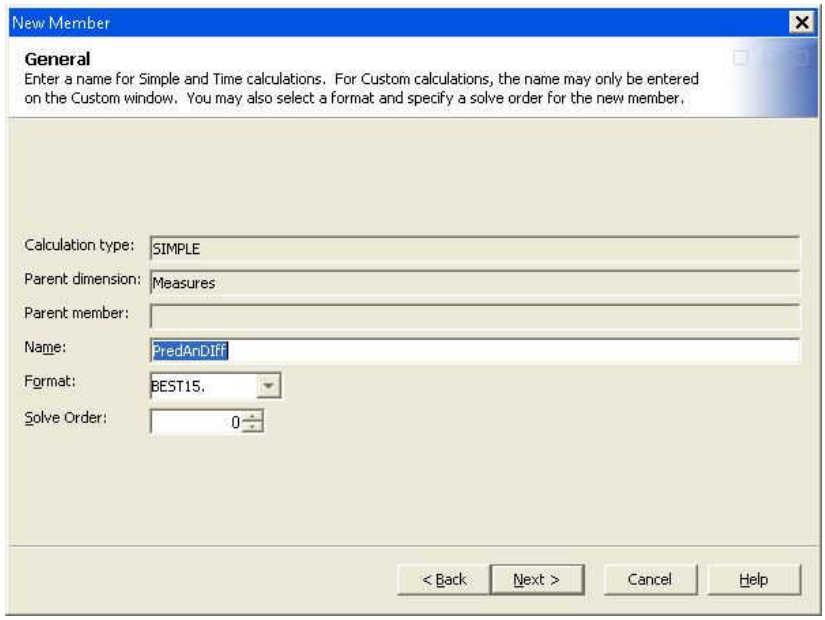
On the Calculation Type page of the New Member wizard, select the type of calculation that you want to create: a simple calculation, time analysis calculation, or a custom calculation. In this example, a simple calculation is created.



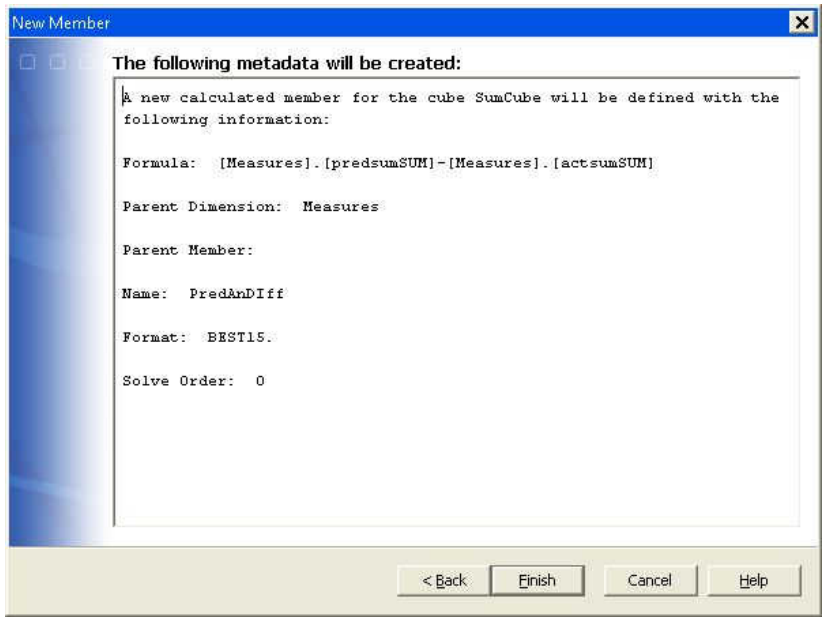
On the Simple Calculations page, select the mathematical calculation that you want to use in the calculated member from the list of radio buttons. Next, select the measures that you want to use with the calculation. In this example, the predicted and actual measures for the cube are used in the formula. It is the difference between these measures that the formula will calculate. Select **Next** when you are finished.



On the General page of the New Member wizard, enter the measure name. In this example, the calculated measure is named PredAnDiff. Select the format and solve order for the measure. When you are finished, click **Next**.

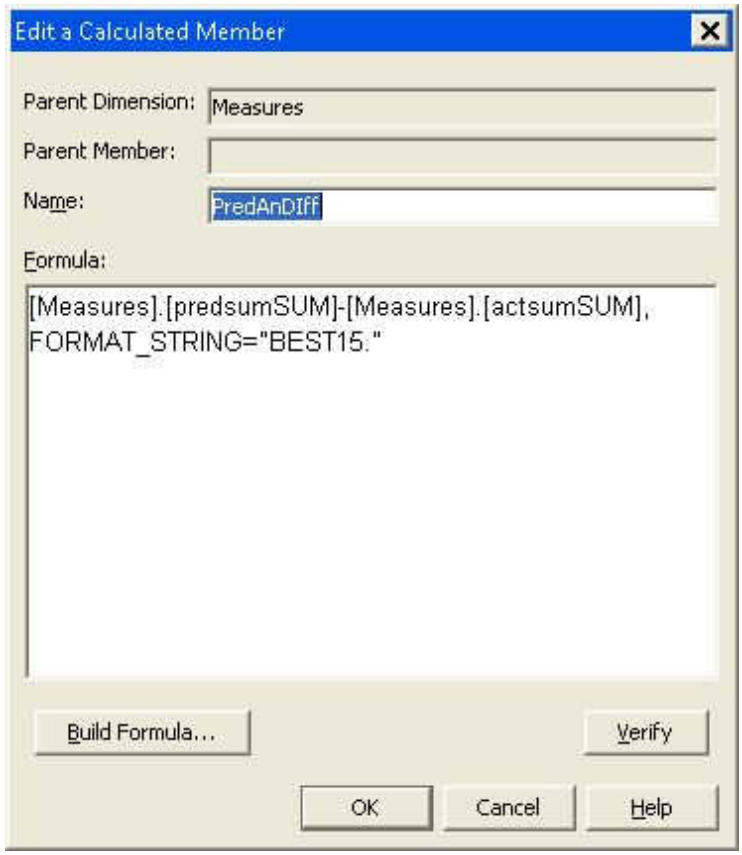


On the Finish page of the New Member wizard, a display panel shows the metadata that is generated for the new calculated member. Review the metadata. When you are finished, click **Finish** to complete the New Member wizard.

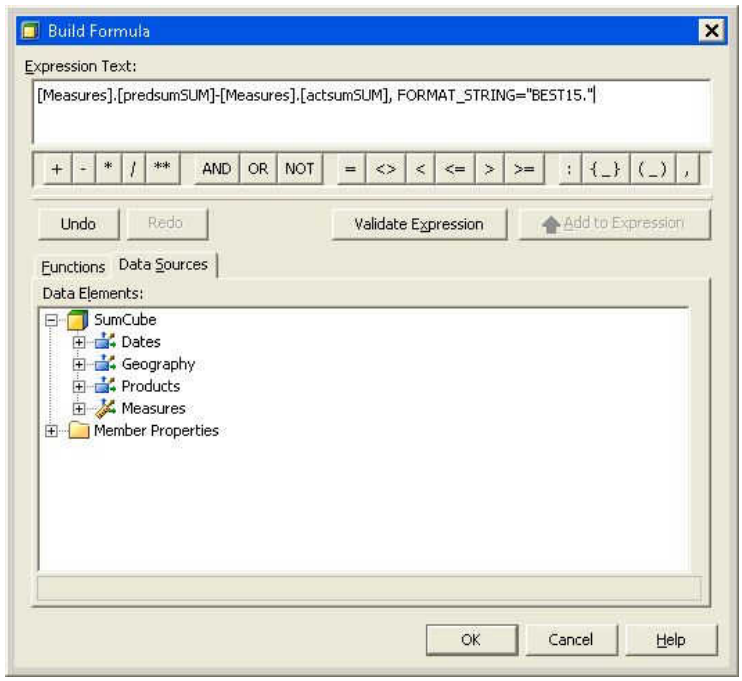


The new calculated member PredAnDiff is now listed in the **Defined calclulated members** list.

If needed, you can now make changes to the PredAnDiff calculated member. On the Calculations for Cube page, select the calculated member and click **Next**. This opens the Edit a Calculated Member page. You can then modify the measure name and formula. Select **Verify** to check the MDX code for errors and validate the code against the OLAP server. If you click **Build Formula**, the Build Formula dialog box appears.



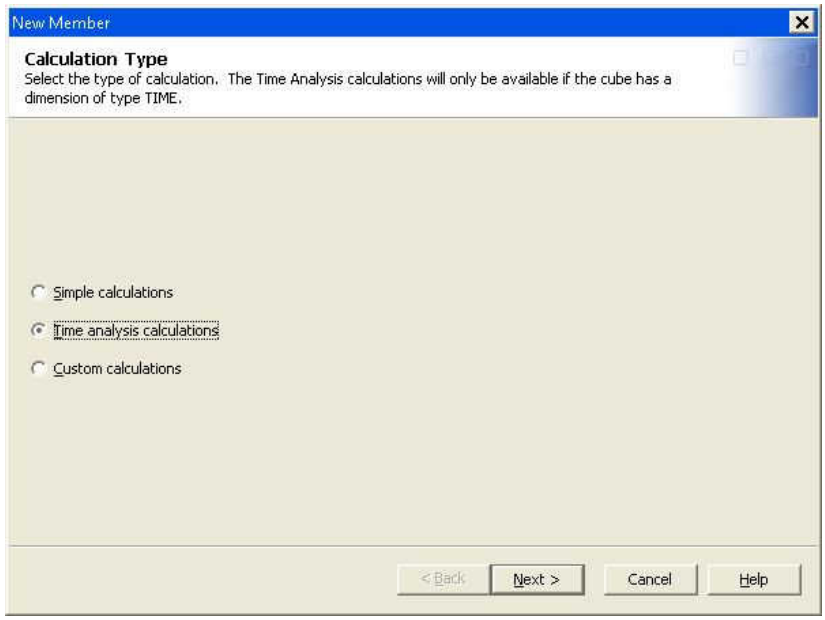
In the Build Formula dialog box, you can modify the MDX code with the help of an expression builder. You can also add MDX functions and Data Source elements to the expression. Select **OK** when you are finished editing the calculated member.



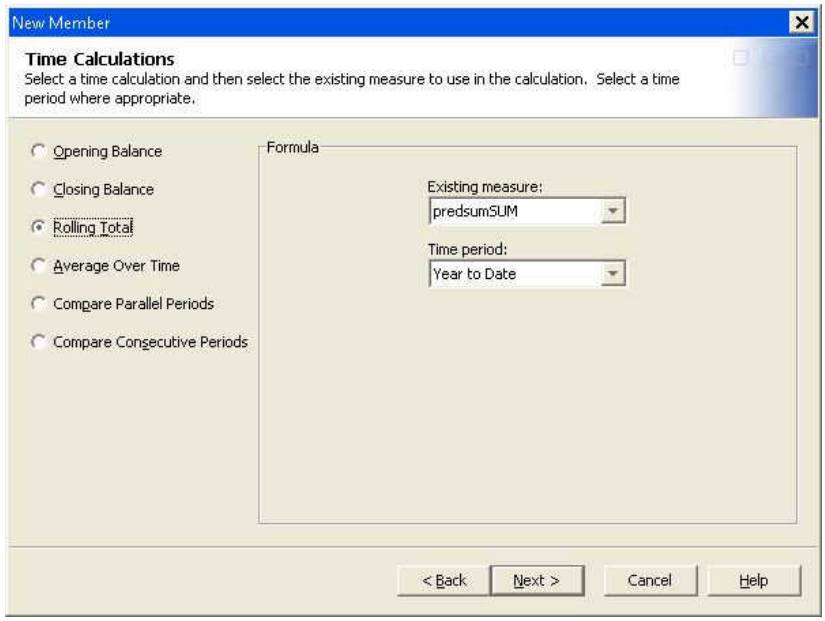
Creating a Time Analysis Calculation

You can also create a Time analysis calculated member with the New Member wizard. On the Calculations for Cube dialog box, click **Add** to launch the New Member wizard and define a new calculated member.

On the Calculation Type page of the New Member wizard, click **Time analysis calculation**. When you are finished, click **Next**.



On the Time Calculations page, select a time calculation. Next, select an existing measure from the **Formula** panel. The **Formula** panel will change depending on the time calculation radio button that you select. The **Existing measure** drop-down list is populated with all measures, including calculated measures. The **Time period** drop-down list is populated with members from the Time dimension. When you are finished, click **Next**.

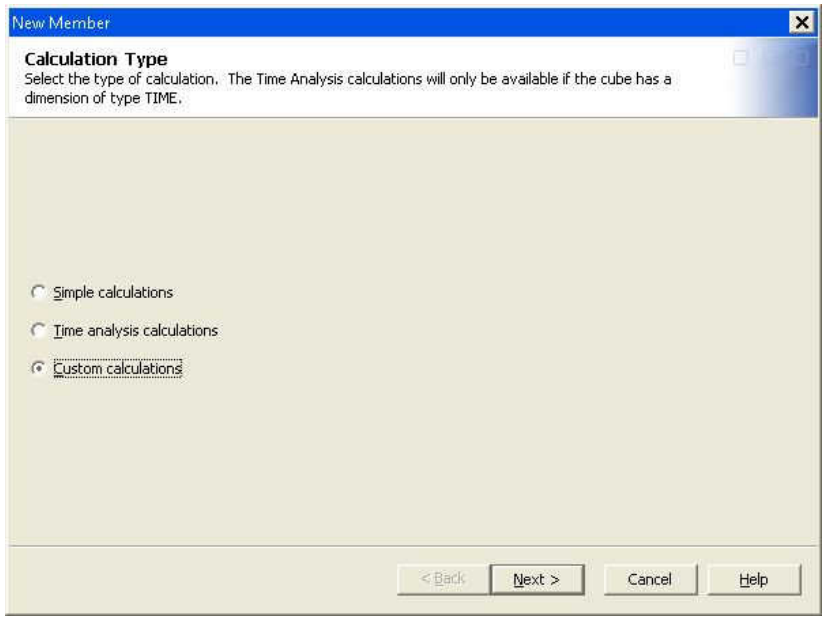


Finish the New Member wizard by completing the **General** and **Finish** pages.

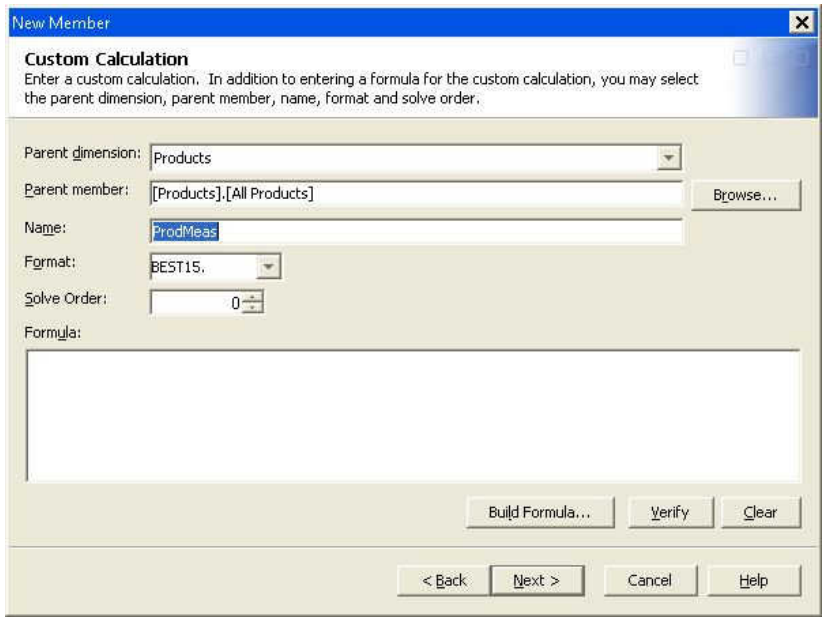
Creating a Custom Calculation

You can also create a custom calculated member with the New Member wizard. On the Calculations for Cube dialog box, click **Add** to launch the New Member wizard and define a new calculated member.

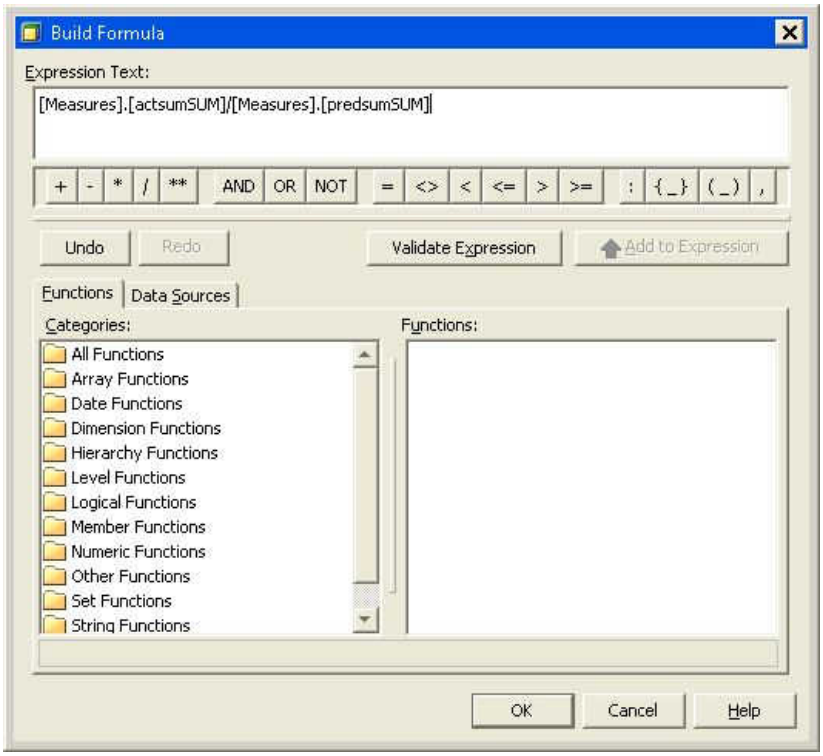
On the Calculation Type page of the New Member wizard, click **Custom calculation**. When you are finished, click **Next**.



On the Custom Calculation page, you can select the parent dimension and parent member for the measure. You can also enter the MDX formula for the measure. Enter the name for the calculated member and select the format and solve order.



If you click **Build Formula**, the Build Formula dialog box appears. In this dialog box, you can modify the MDX code with the help of an expression builder. You can also add MDX functions and data source elements to the expression.



On the Custom Calculation page, click **Verify** to check any MDX code that you might have added for errors. This validates the code against the OLAP server. Select **OK** when you are finished editing the calculated member. Finish the New Member wizard by completing the **General** and **Finish** panels.

Setting Member–Level Permissions

SAS security enables you to set authorization permissions on a cube and the various components of a cube. You can apply member-specific filters to cube data by using MDX expression filters known as permission conditions. Permission conditions limit access to a cube dimension so that only designated portions of the data are visible to a user or group of users. With SAS OLAP data, permission conditions impose only explicit grants of the Read permission and can be specified only on dimension objects. You can add member authorizations to a cube dimension from either SAS OLAP Cube Studio or SAS Management Console. In this example, a retail company has a SAS OLAP cube that contains sales data. They need to apply Read restrictions to members of a dimension that contains data for the method of payment for merchandise.

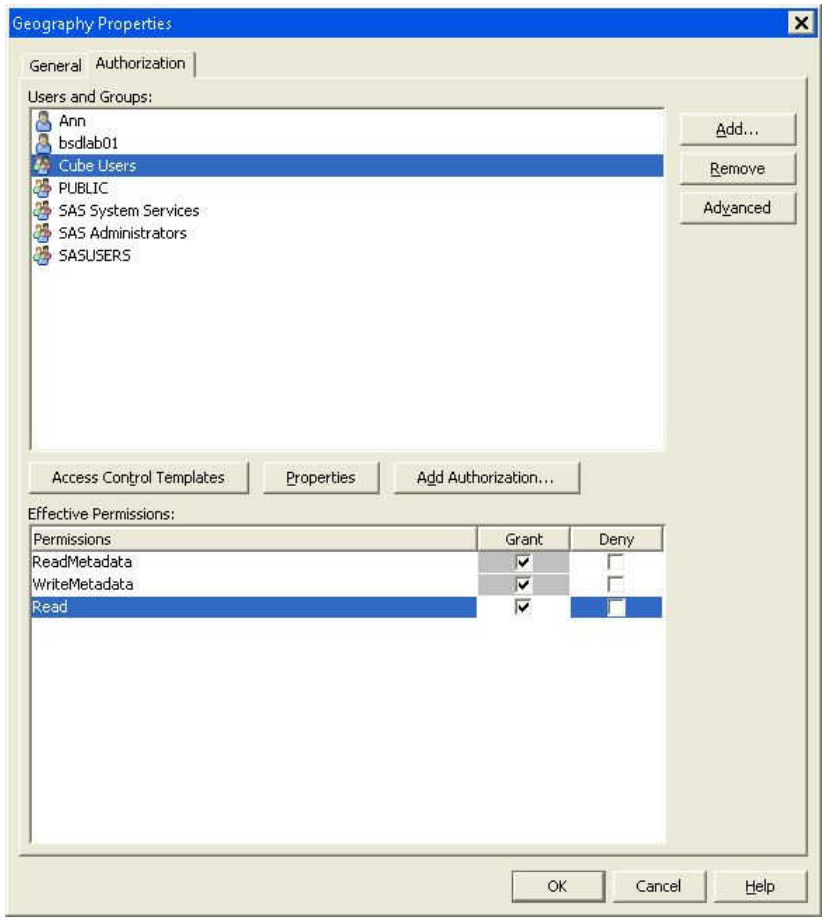
To select a dimension from within SAS OLAP Cube Studio, select a cube in the tree view and drill down to a dimension. To select a dimension in SAS Management Console:

1. Select **Authorization Manager** ⇨ **By Type** ⇨ **Dimension** and drill-down to a dimension.
2. Right-click the dimension and click **Properties**.

3 In the dimension's Properties dialog box, select the **Authorization** tab, as shown in the following display. Select (or add) the user or group whose Read access you want to limit. In this example, the **PUBLIC** group is restricted.

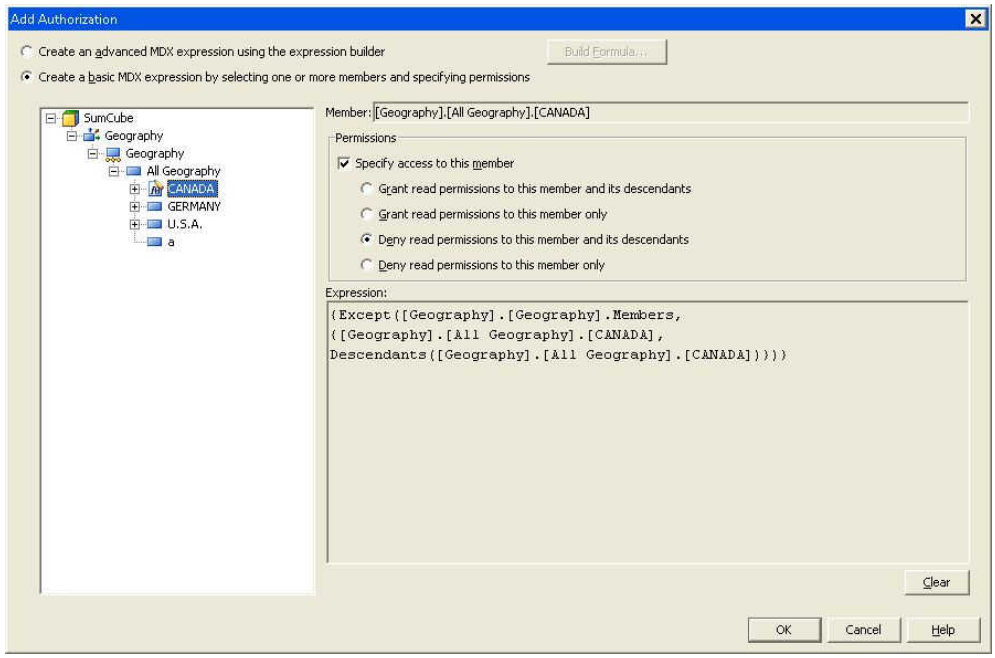
4 In the **Effective Permissions** list, add an explicit grant of the **Read** permission for that user or group. If the selected user or group does not already have a permission condition defined, the **Add Authorization** button is now enabled.

5 Click **Add Authorization** to open the Add Authorization dialog box.

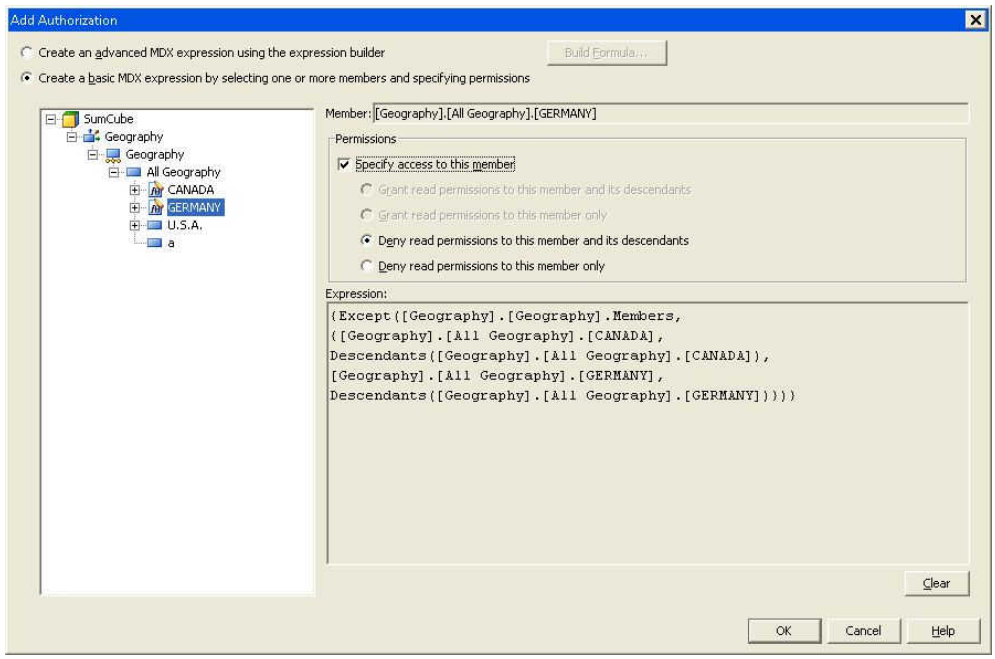


On the Add Authorization dialog box, you can create an MDX expression by specifying members and permissions for the permission condition. You can choose to create a basic expression or an advanced expression for a member. By default, the option **Create a basic MDX expression by selecting one or more members and specifying permissions** is selected when you open the Add Authorization dialog box. In this example, the Geography dimension contains data for different countries. The members Canada and Germany are restricted from view for the CubeUsers group. Drill down on the dimension in the list and select the Canada member. Set the access for the Canada member.

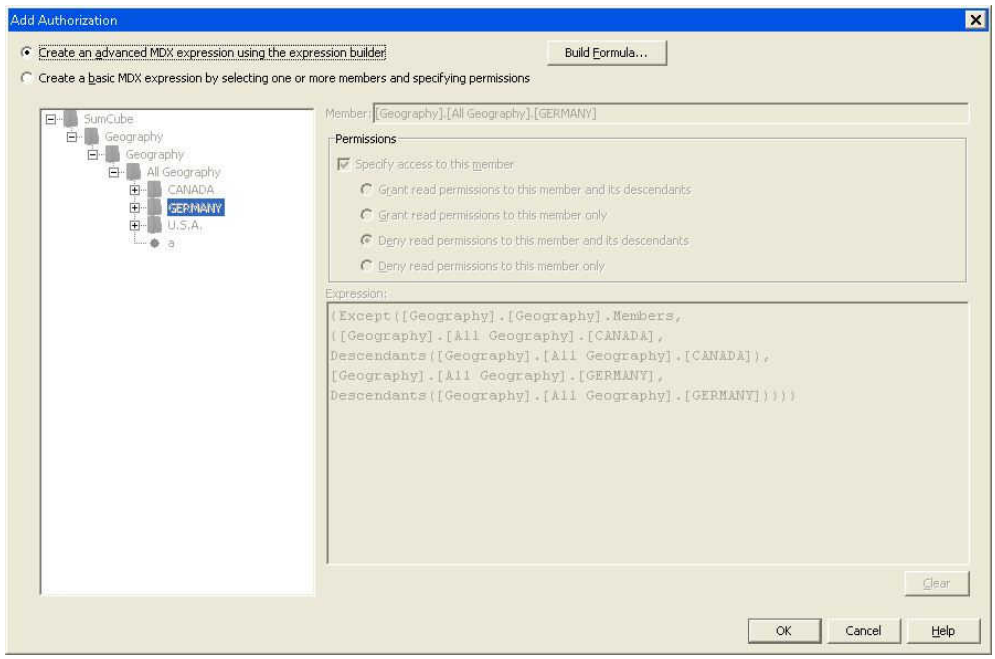
After you have selected the basic expression option and a member, the MDX code for the permission condition will appear in the **Member** text field. You can now select the option **Specify access to this member**. This enables you to apply a permission option to the currently selected member or a member and its descendants. You must select this option to activate one of the grant or deny permissions options. Click the **Deny read permissions to this member and its descendants** option. The MDX code for the permission condition is displayed in the **Expression** text box.



Repeat the process for the Germany member. The MDX code for Germany is appended to the MDX code for Canada member in the **Expression** text box.

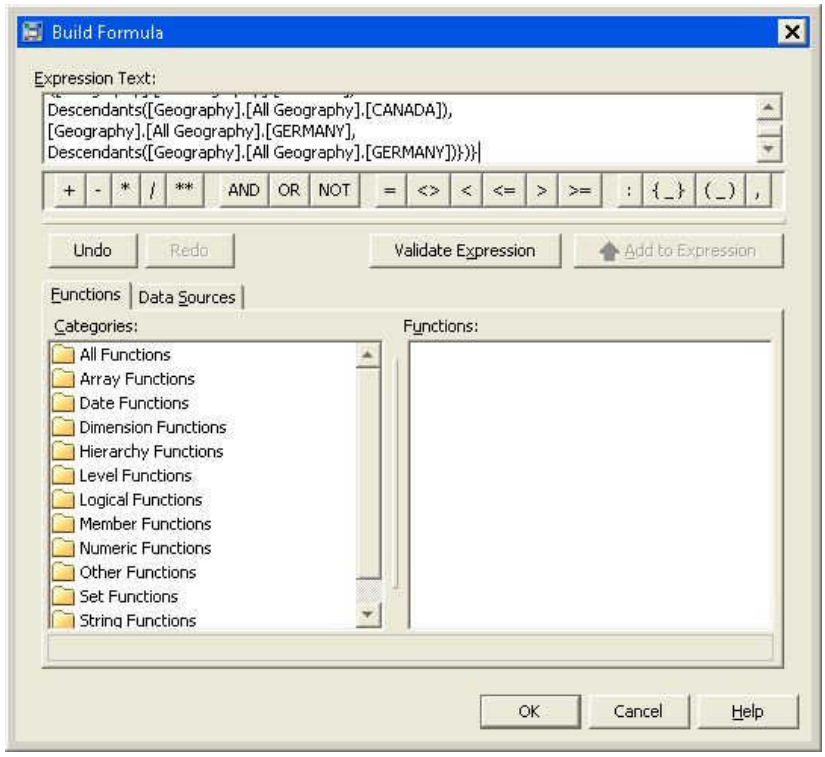


You can also choose to manually create an MDX expression filter. In the Add Authorization dialog box, select the option **Create an advanced expression using the expression builder**. The **Build Formula** button is now active. Click **Build Formula**. The Build Formula dialog box appears.

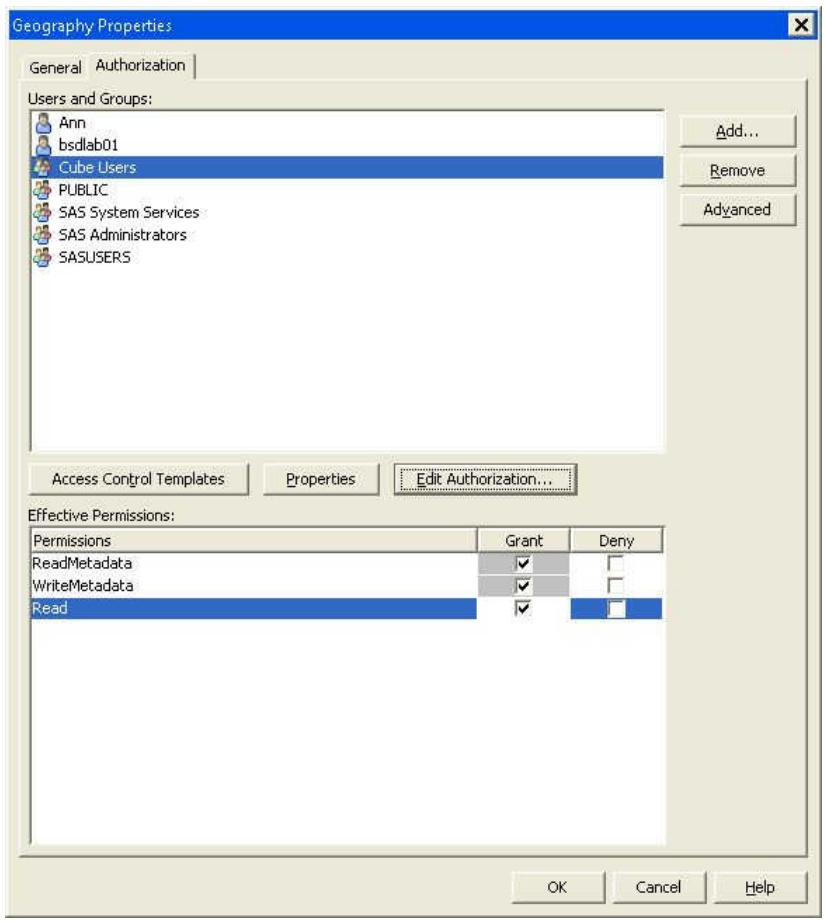


Note: If you have already specified a basic permission condition that denies access to a member, and you select the **Create an advanced expression using the expression builder** option, a message appears that asks: “Do you want your current expression copied to the expression builder?” See the following display. Select **Yes** or **No**. You can then click **Build Formula**.

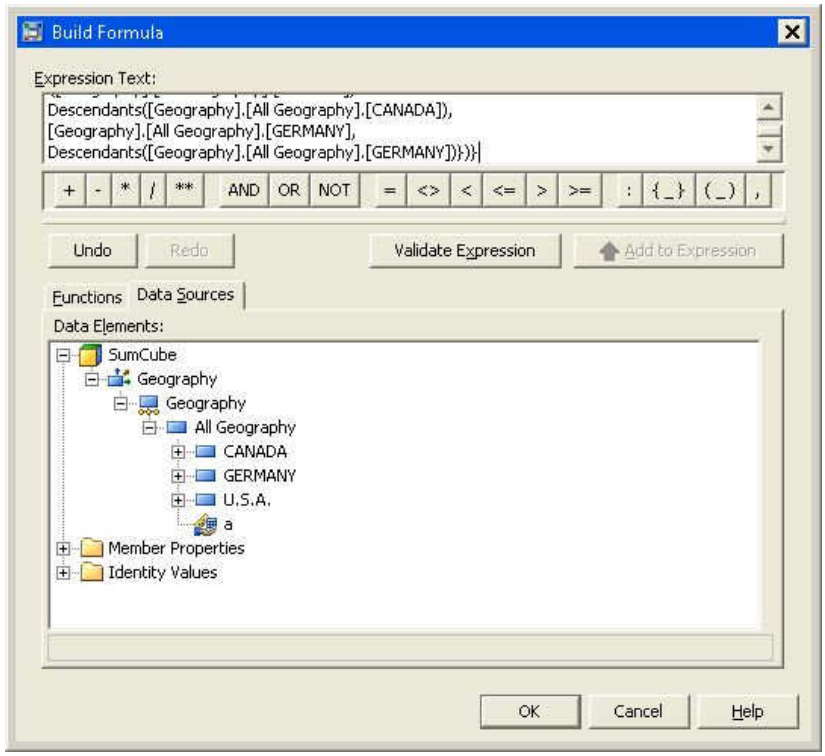
In the Build Formula dialog box, you can create an MDX filter and observe the MDX expression as you build it. Use the logical operators to specify multiple clauses in your MDX expression in the **Expression Text** text box. Use the **Functions** tab to add MDX functions to your expression. Use the **Data Sources** tab to browse through the dimensions and hierarchies in your cube and select the members that require access control. You can use the **Add to expression** button to add your selections to the **Expression Text** text box. You can also check the accuracy of the expression that you are building by selecting the **Validate Expression** button. When you are finished, click **OK**. You will return to the Add Authorization dialog box. Select **OK** again to save the permission condition and return to the Properties dialog box.



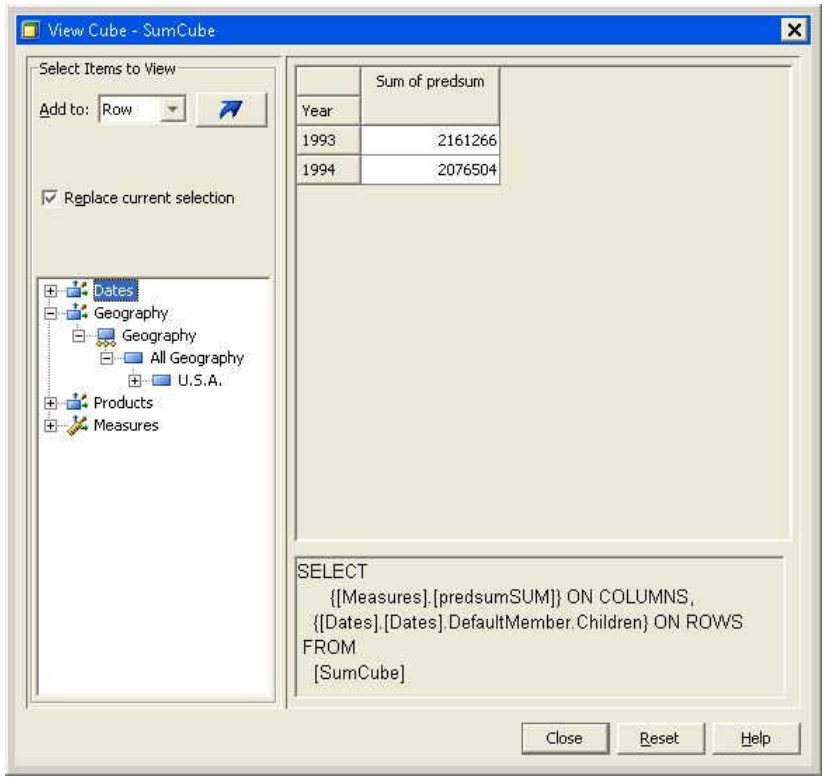
After you have defined permission conditions for the dimension, the label for the **Add Authorization** button changes to **Edit Authorization**.



If you later click **Edit Authorization** to edit a permission condition, the Build Formula dialog box will open. You can then make any needed changes to the MDX code. Select **OK** to save the permission condition and return to the Properties dialog box.



After you have applied the permission conditions, you can validate the Read restrictions with the View Cube function in SAS OLAP Cube Studio. In SAS OLAP Cube Studio, select the SALES cube and click **View Cube** from the **Actions** menu. You can see in the following display that the members Canada and Germany are not available for view on the Geography dimension.



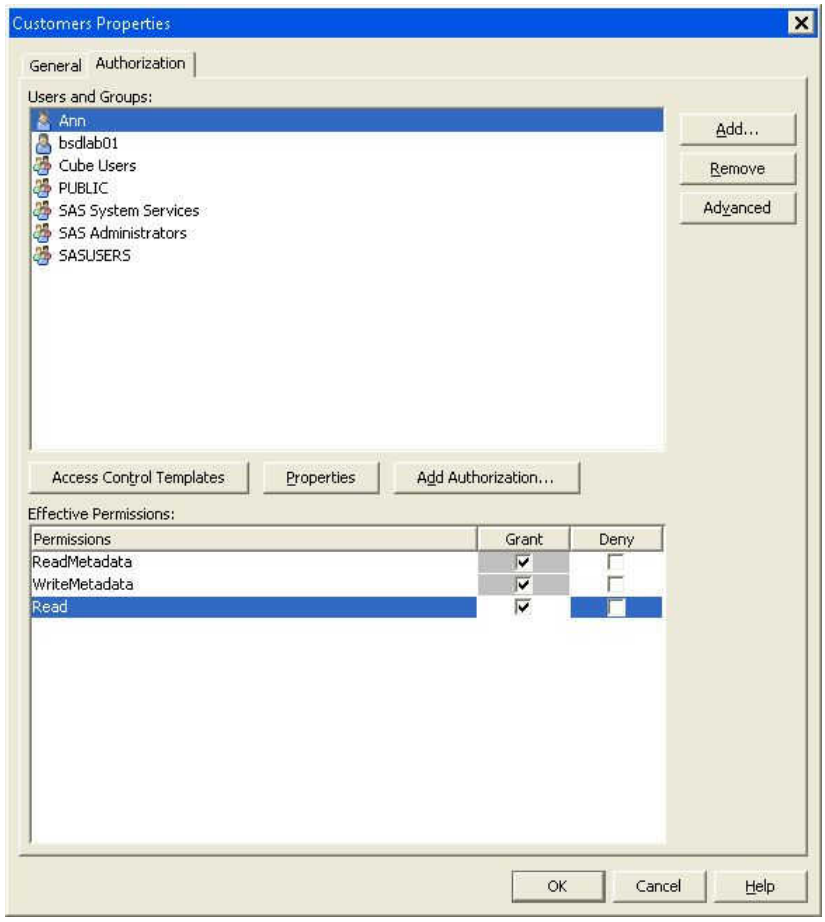
You can also set permission conditions for a dimension by using PROC OLAP and MDX expressions. See the SECURITY_SUBSET option for the PROC OLAP statement and “SAS OLAP Security Totals and Permission Conditions” topic in the *SAS OLAP Server: MDX Guide*.

Setting Identity-Driven Security

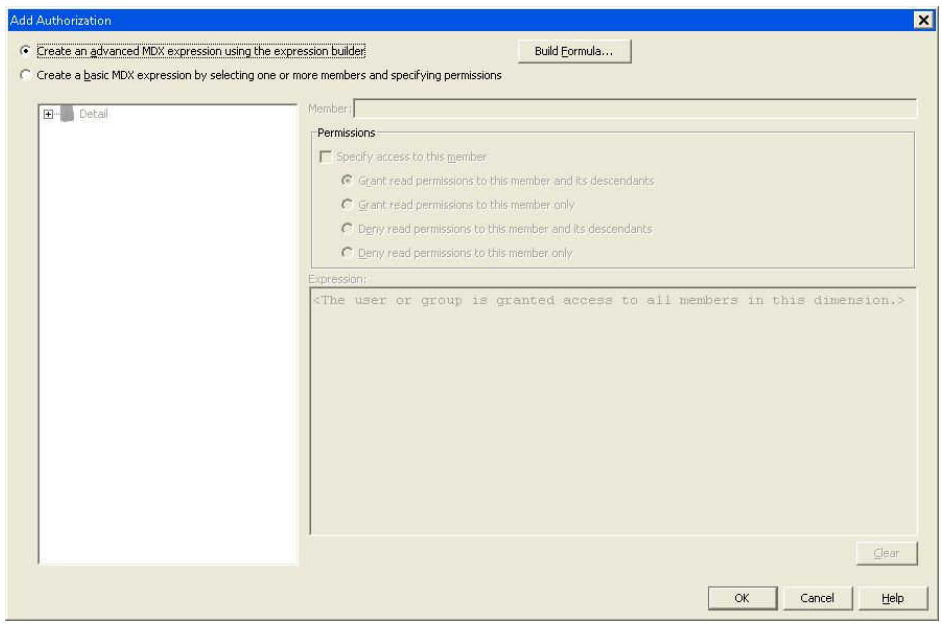
It is sometimes necessary to substitute identity values in a permission condition to further refine member-level security. Identity-specific values are dynamically derived according to the user ID with which a client is authenticated. Those values are then used to filter the target data. The identity-specific values are derived from identity-driven properties that are stored in the metadata repository for each user and group. You can set an identity driven authorization using the Member Authorization expression builder.

1. Select **Authorization Manager** ⇨ **By Type** ⇨ **Dimension** and drill down to a dimension.
2. Right-click the dimension and select **Properties**.
3. In the dimension's Properties dialog box, select the **Authorization** tab, as shown in the following display. Select (or add) the user or group whose Read access you want to limit. In this example, the **PUBLIC** group is restricted.
4. In the **Effective Permissions** list, add an explicit grant of the **Read** permission for that user or group. If the selected user or group does not already have a permission condition defined, the **Add Authorization** button is now enabled.

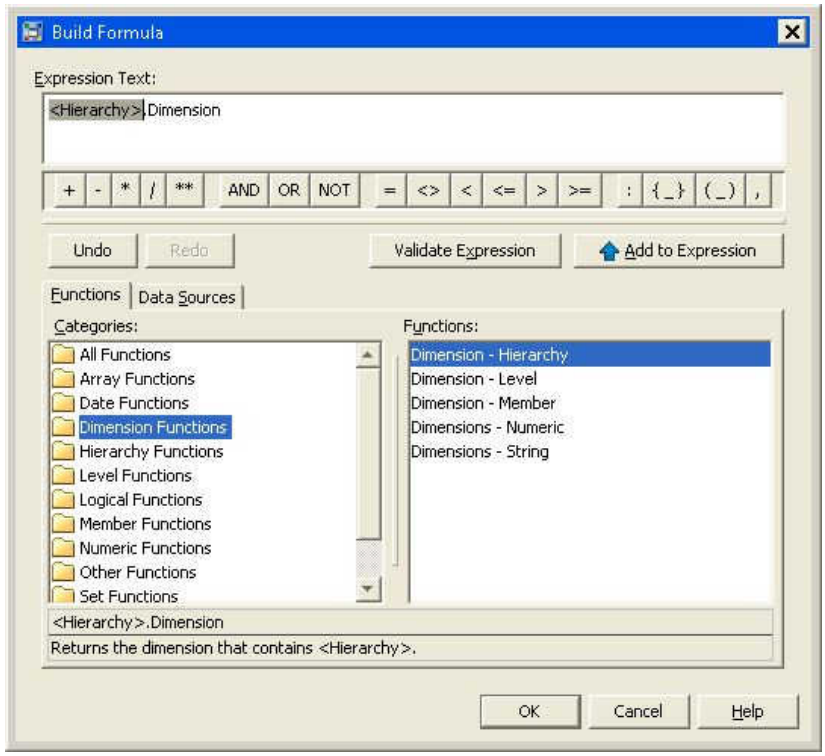
5 Click **Add Authorization** to open the Add Authorization dialog box.



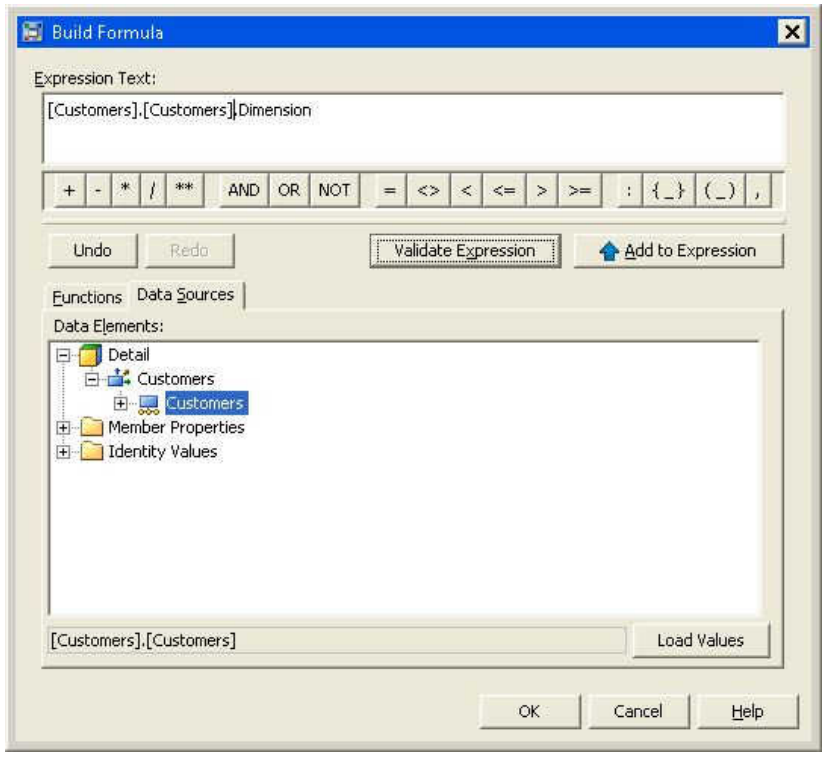
In the Add Authorization dialog box, select the option **Create an advanced MDX expression using the expression builder** option. You can then click **Build Formula**. This opens the Build Formula dialog box.



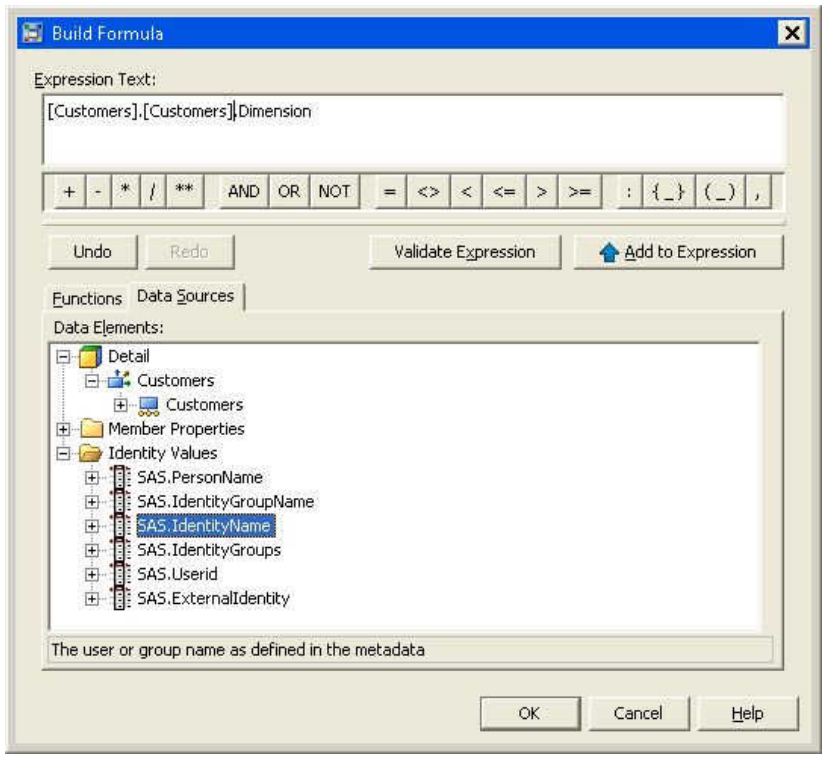
In the Build Formula dialog box, you can create an MDX filter and observe the MDX expression as you build it. Use the logical operators to specify multiple clauses in your MDX expression in the **Expression Text** list. Use the **Functions** tab to add MDX functions to your expression. Use the **Insert** button to add your selections to the **Expression Text** list.



Use the **Data Sources** tab to browse through the dimensions and hierarchies in your cube and select the members that require access control. Use the **Add to Expression** button to add your selections to the **Expression Text** text field. You can also check the accuracy of the expression that you are building by selecting the **Validate Expression** button.



To add identity values to the expression, click the **Identity Values** folder on the **Data Sources** tab. Select an identity value from the list. Use the **Add to Expression** button to add your selections to the **Expression Text** text field.



Here is a list of possible identity values:

SAS.ExternalIdentity
This property translates to optional, site-specific values such as Employee ID. Those values are not automatically stored in the metadata repository and need to be loaded and maintained.

SAS.IdentityGroupName
This property resolves to the name of the requesting group identity (for example, Portal Admins Group).

SAS.PersonName
This property resolves to the name of the requesting user identity (for example, SAS Demo User).

SAS.IdentityName
This property returns the name of either the requesting group identity or the requesting user identity, depending on whether the user ID is a group login or a personal login.

SAS.Userid
This property translates to the authenticated user ID, normalized to one of the uppercase formats USERID or USERID@DOMAIN (for example, SASDEMO@LXXXXX).

SAS.IdentityGroups
This property resolves to the names of the groups of which a user is a member.

When you are finished, click **OK**. You will return to the Add Authorization dialog box. Select **OK** again to save the permission condition and return to the Properties dialog box.

See the topics "Securing Cubes" on page 119 and "Identity-Driven Properties" on page 120 for more information.

Viewing a Cube in SAS OLAP Cube Studio

Overview

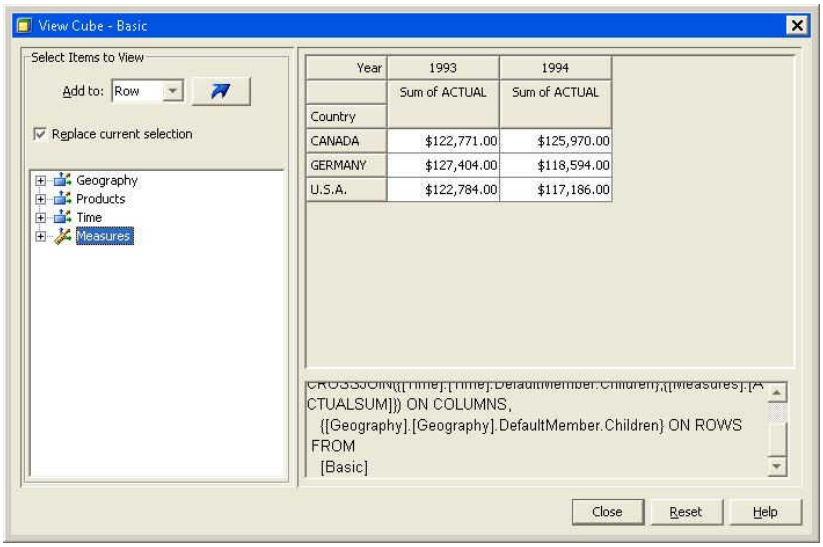
After you have built a cube in SAS OLAP Cube Studio, you can examine the contents of the cube with the View Cube function. The View Cube function is available from the **Actions** menu and from the cube context menu. The View Cube function is available for use only with physically built cubes. In addition, when you select the View Cube function, a connection to a SAS OLAP Server must be made.

When the View Cube function is selected, the View Cube dialog box is loaded and a multidimensional view of the cube data is displayed in a table. By default, the first dimension (default hierarchy) of the cube is displayed on the row axis and the default measure is displayed on the column axis of the table. The dimensions that are not assigned to a row or column are automatically assigned to the slicer axis. The default member or ALL member is used to create the slicer filter. You can then drill into the dimensions for the cube and see figures for specific levels of the cube.

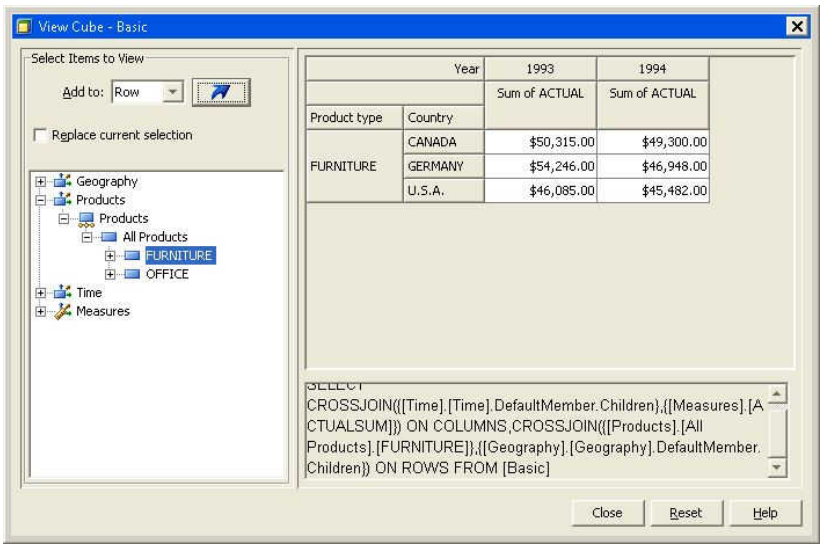
In the View Cube dialog box, you can choose to add level selections, or you can select the check box **Replace current selection** to replace the existing levels.

Adding Level Data to a Cube View

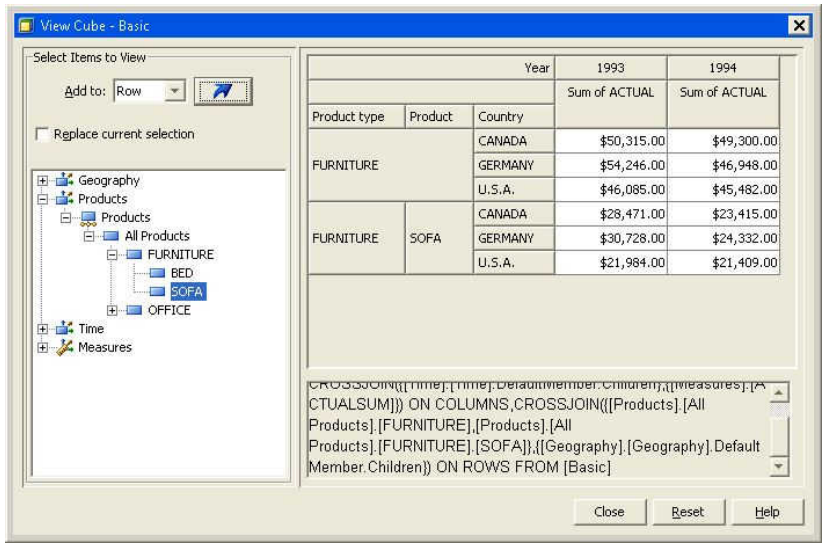
In this example, the option **Replace current selection** is not selected. A retailer is analyzing sales figures for outlets in various countries. In the initial cube view, the column axis displays the default measure for the **Sum of ACTUAL** sales. Displayed on the row axis is the default hierarchy for the **Geo** (geography) dimension.



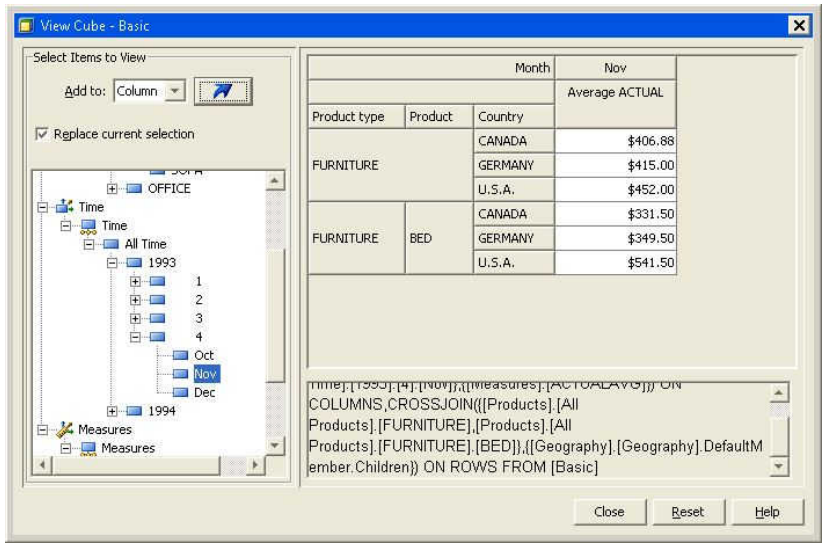
In the following display, the row axis view is filtered on the **Products** dimension to the **FURNITURE** level. Furniture-specific sales numbers are added to the existing row axis. The column axis is not modified and is still filtered on the **Sum of ACTUAL** sales.



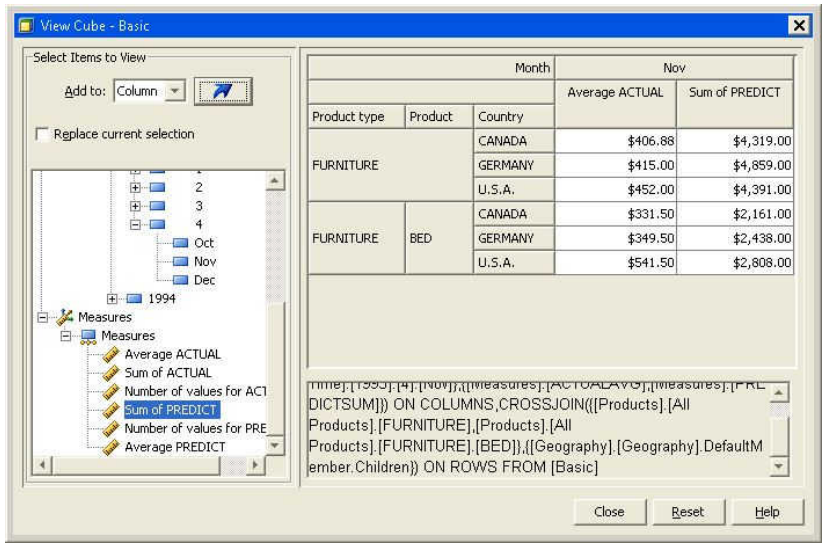
The following display shows continued filtering on the **FURNITURE** level down to the sales figures for sofas only.



You can also filter on the column values. In this display, the **Time** dimension is filtered down to the sales figures for the month of November. In addition, the **Products** dimension has been filtered on beds for the row axis.

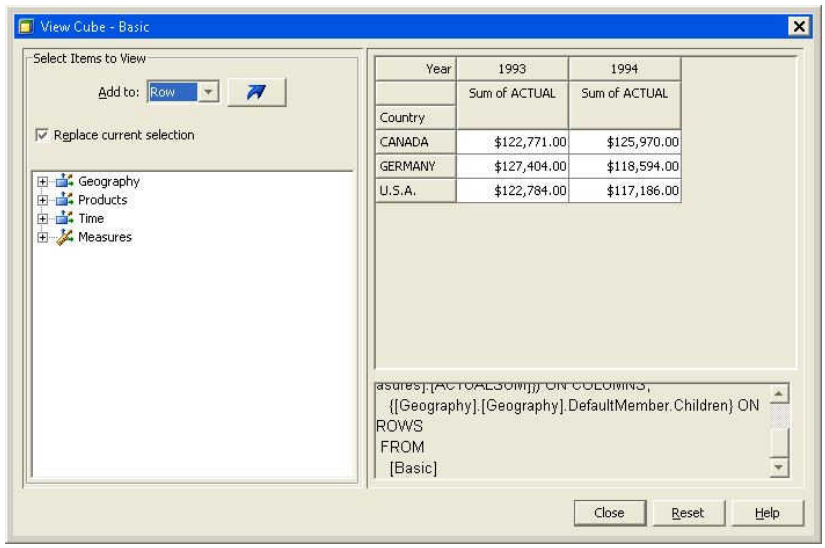


In this display, the measure **Sum of PREDICTED** sales is added to the column axis. You can now compare the actual versus the predicted sales totals.

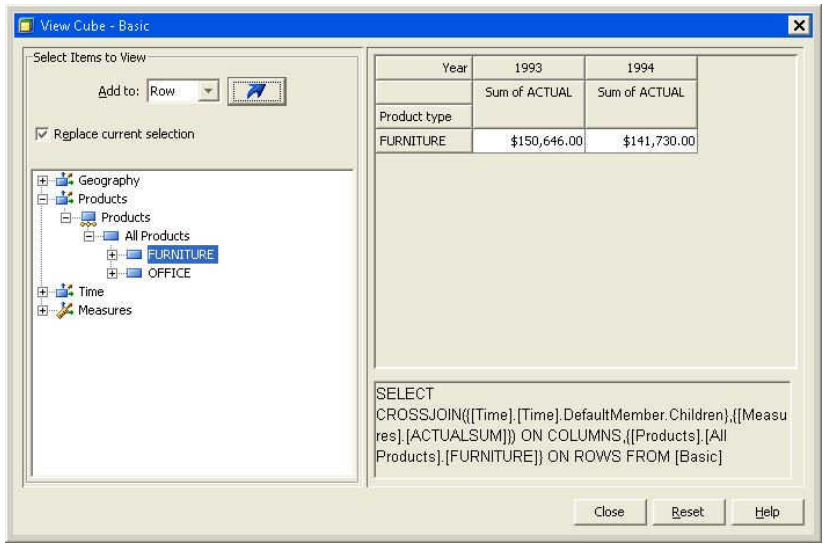


Replacing Level Data on a Cube View

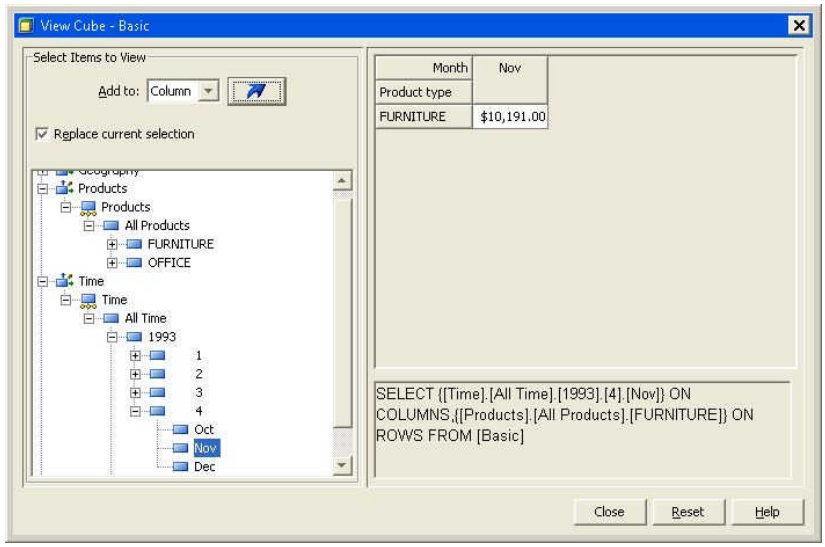
In this example, the option **Replace current selection** is checked. A retailer is analyzing sales figures for outlets in various countries. In the initial cube view, the column axis displays the default measure for the **Sum of ACTUAL** sales. Displayed on the row axis is the default hierarchy for the **Geo** (geography) dimension.



In the following display, the row axis view is filtered on the **Products** dimension to the **FURNITURE** level. Because the **Replace current selection** check box is selected, the original, default row selections were replaced. The column axis is not modified and is still filtered on the **Sum of ACTUAL** sales.



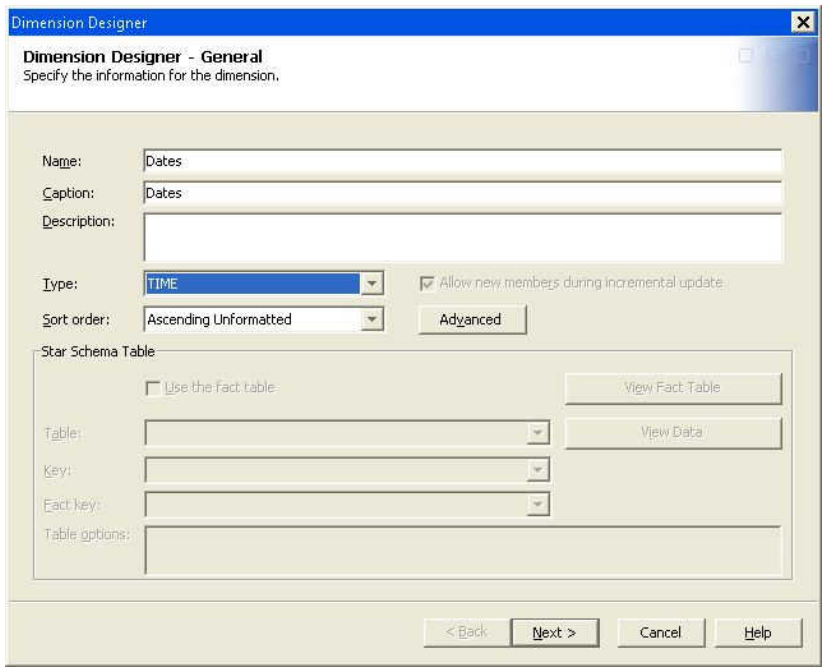
You can also filter on the column values. In this display, the **Time** dimension is filtered down to the sales figures for the month of November. Because the **Replace current selection** check box is selected, the original, default column selections were replaced.



Creating a TIME Dimension in SAS OLAP Cube Studio

When you create a cube in SAS OLAP Cube Studio, the Cube Designer wizard enables you to create cube dimensions and add the needed levels and hierarchies to those dimensions. For time-specific dimensions, you can select from a list of supplied time hierarchies. The time hierarchies will help you build the dimension and auto-populate the levels.

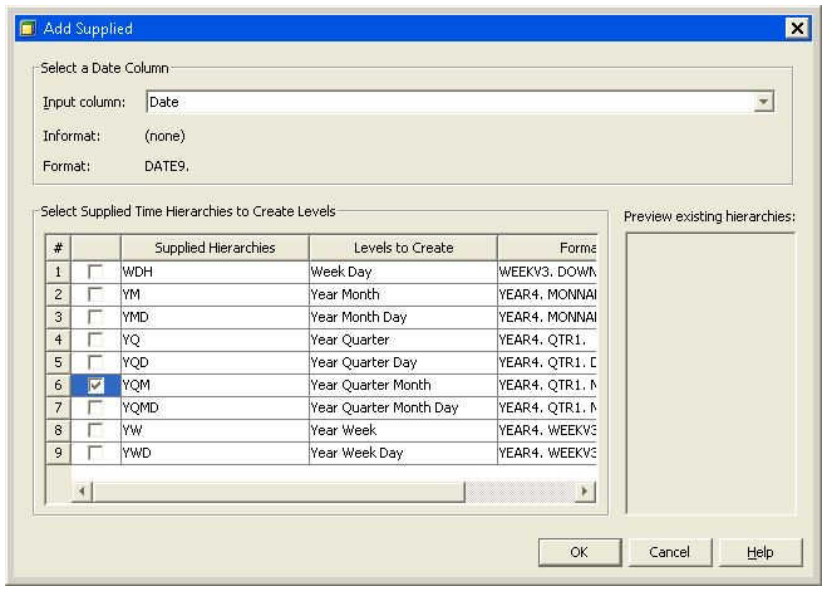
Open the Cube Designer wizard and select the input data for the cube. Select **Add** on the Cube Designer – Dimensions page. This opens the Dimension Designer – General page. When you create a time dimension, you must select the **TIME** dimension type. See the following display.



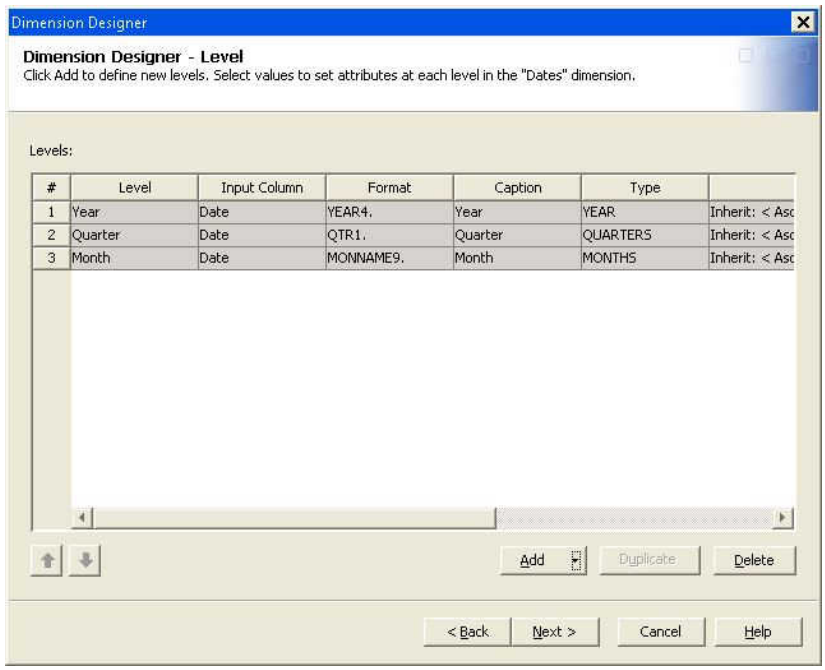
Enter the remaining information for the dimension and click **Next**. This opens the Dimension Designer– Level page. On the Dimension Designer – Level page, the **Add** button becomes a drop-down list of options. The **Add levels** and **Add supplied time hierarchies** options are now available for selection.

Note: The Add (levels) function is always available for selection regardless of the type of dimension that you are creating. When you specify a **TIME**-type dimension, the **Add** button is converted to a drop-down list where you can select the function that you want to use.

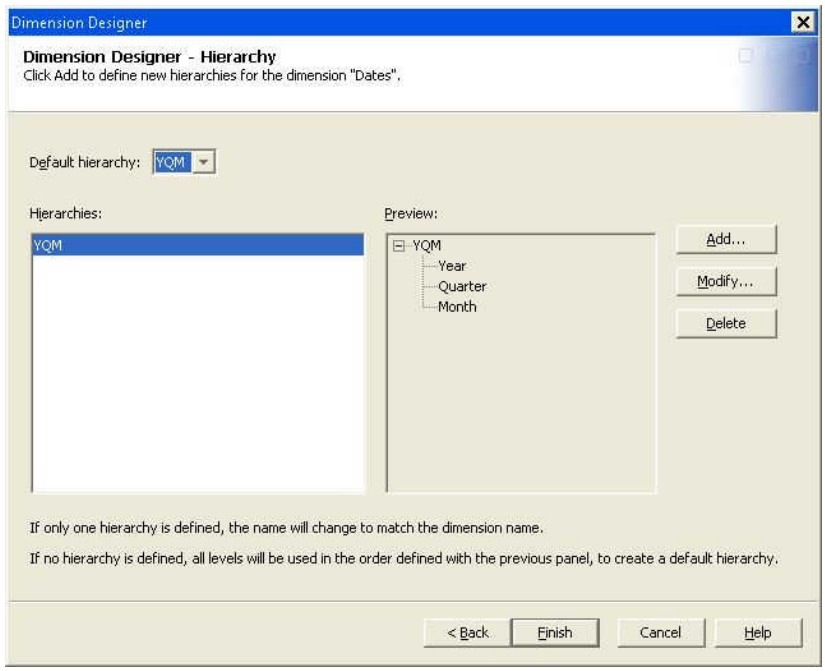
Select the **Add supplied time hierarchies** option. On the Add Supplied page, the **Input column** drop-down list displays the available numeric values that have a date format applied to the column. This will default to the **Date** value if it is found in the data source list of columns. The **Format** and **Informat** values change, depending on the value selected in the **Input column** drop-down list. Select from the list of supplied time hierarchies to create the time levels. See the following display.



Selecting **OK** closes the dialog box and updates the Dimension Designer – Level table with new levels based on your selection. One level is created for each selected time period, even if it appears in multiple supplied hierarchies. You can then define properties such as the sort order and description for the levels that you have selected. See the following display.



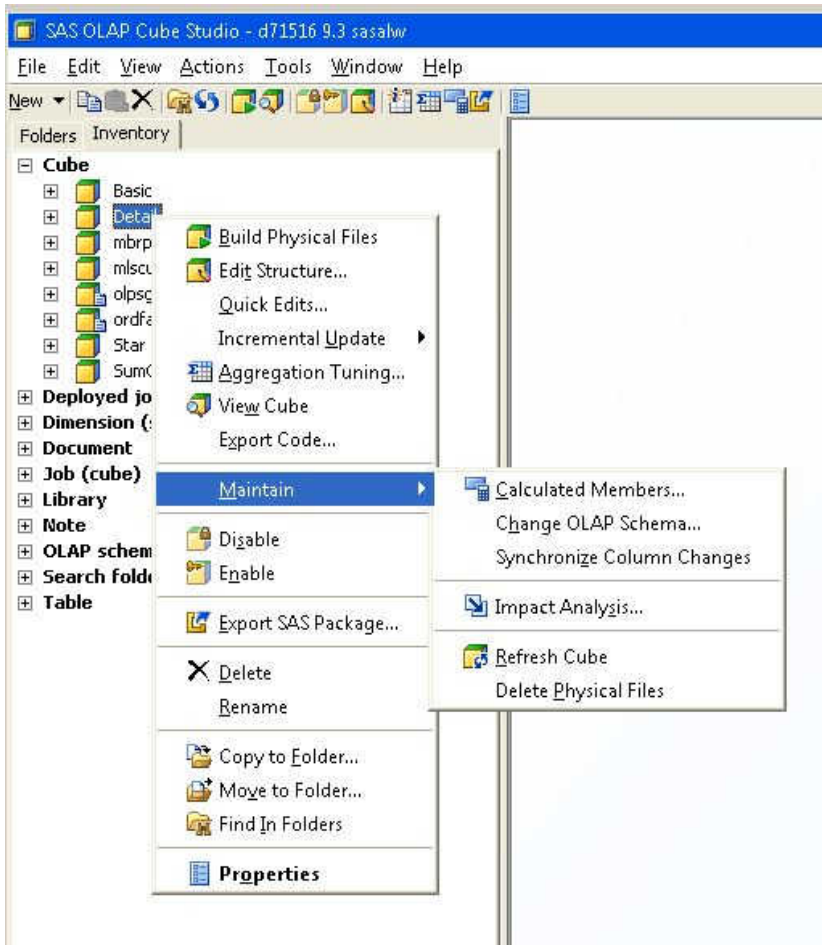
The hierarchy or hierarchies that are selected on the Add Supplied page are listed in the **Hierarchies** panel on the Dimension Designer – Hierarchies page. If there is only one hierarchy, as with this example, the hierarchy name is changed to match the dimension name. See the following display.



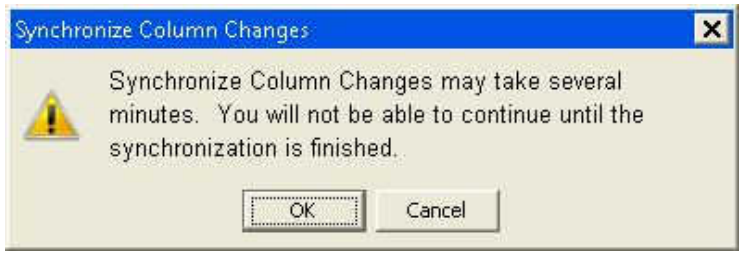
Synchronizing Column Changes

The Synchronize Column Changes function enables you to synchronize a cube when the input table for an existing cube has encountered a column name change. This function finds the name differences between the cube and its input table and changes the hierarchy level names to match the input table column names. The Synchronize Levels function obtains the cube metadata from the metadata repository and compares the names between the input table and the cube hierarchy. If a discrepancy is found, a new cube file and definition are created with the new level name. The level name of the existing cube is then updated to reflect the new column name. The Synchronize Levels function is available for a cube if the cube physically exists and you have the WriteMetadata permission for that cube.

In SAS OLAP Cube Studio, select a cube in the tree view and right-click. On the context menu, select **Maintain** ⇨ **Synchronize Column Changes**.



The Synchronize Column Changes message dialog box appears. Select **OK** to continue with the synchronize function. Select **Cancel** to exit the function.



You can also use the SYNCHRONIZE_COLUMNS option if you are synchronizing the column changes with a PROC OLAP statement. See the SYNCHRONIZE_COLUMNS option in the PROC OLAP statement for more information.

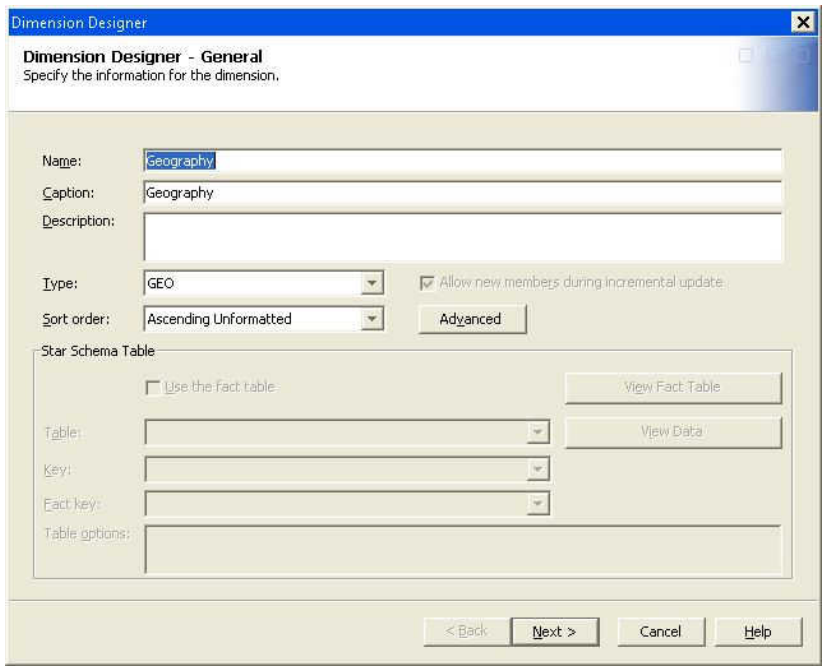
Specifying an Esri GIS Map for a Cube Dimension

When you create a cube in SAS OLAP Cube Studio, you can include the connection information for Esri GIS mapping services. This GIS information can then be read by the SAS OLAP Server and returned during a cube query. In order to access Esri mapping data for a cube, you must have defined an Esri map server in SAS Management Console and have installed the Esri plug-in to SAS Management Console.

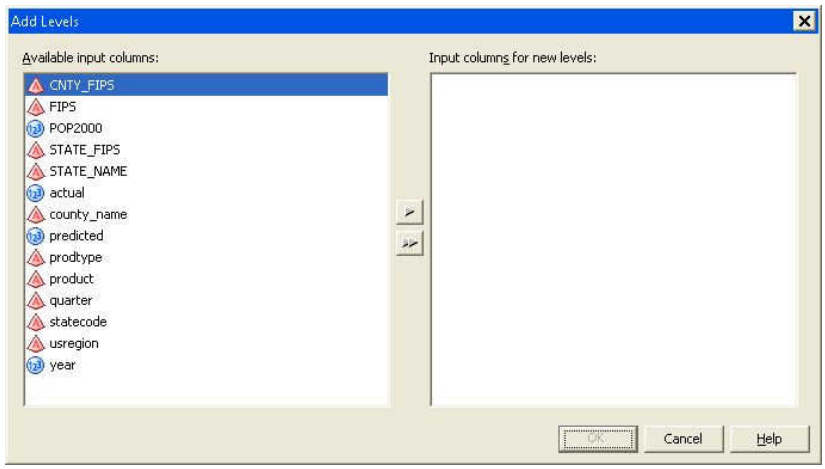
You can add Esri mapping connections to a cube while you are creating the cube or at a later point when you are editing the cube. The Cube Designer wizard in SAS OLAP Cube Studio enables you to define a GIS-specific dimension for a cube and specify the Esri map server that you connect to. In this example, a dimension named US Geography is defined.

Note: For more information about defining Esri GIS mapping connections for SAS OLAP cubes, see “Specifying GIS Map Information for a Dimension” on page 68 .

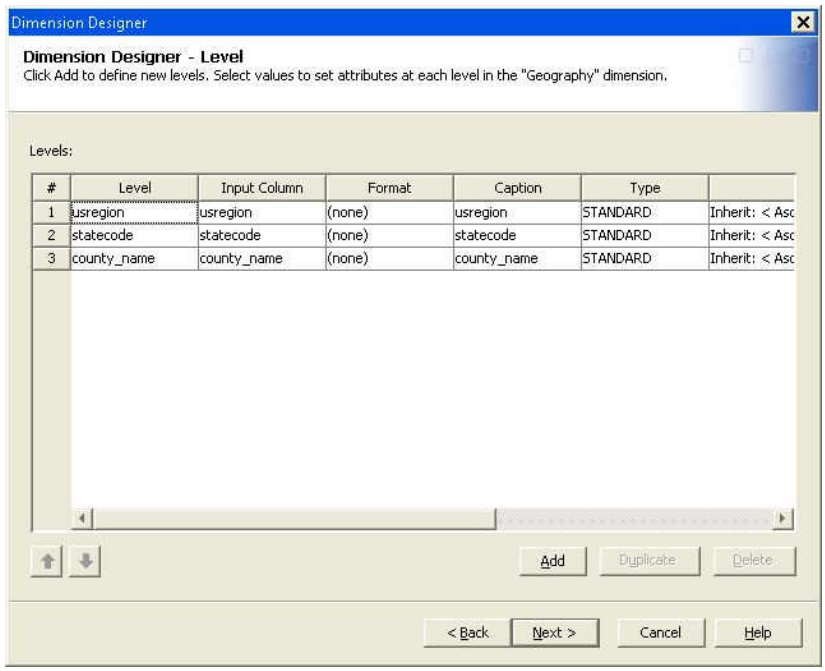
On the Dimension Designer – General page of the Cube Designer wizard, enter basic information for the dimension. For the dimension **Type**, click **GEO**. This will identify the dimension as a geography-specific dimension for Esri mapping. You can have only one GEO-type dimension per cube. When you click **GEO**, the **Specify Map** button is enabled on the Cube Designer – Dimensions page.



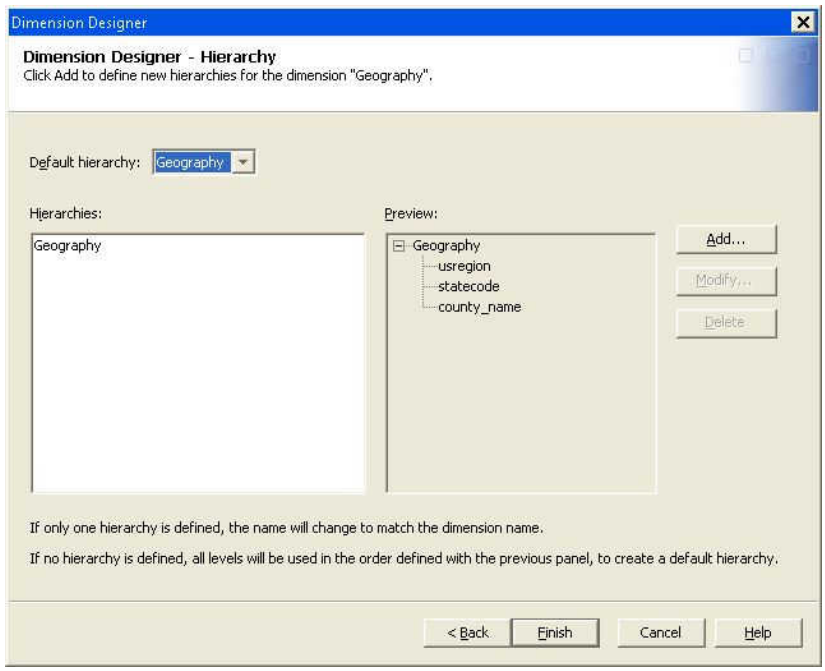
Continue defining the GEO-type dimension. Select the levels for the dimension with the Add Levels dialog box.



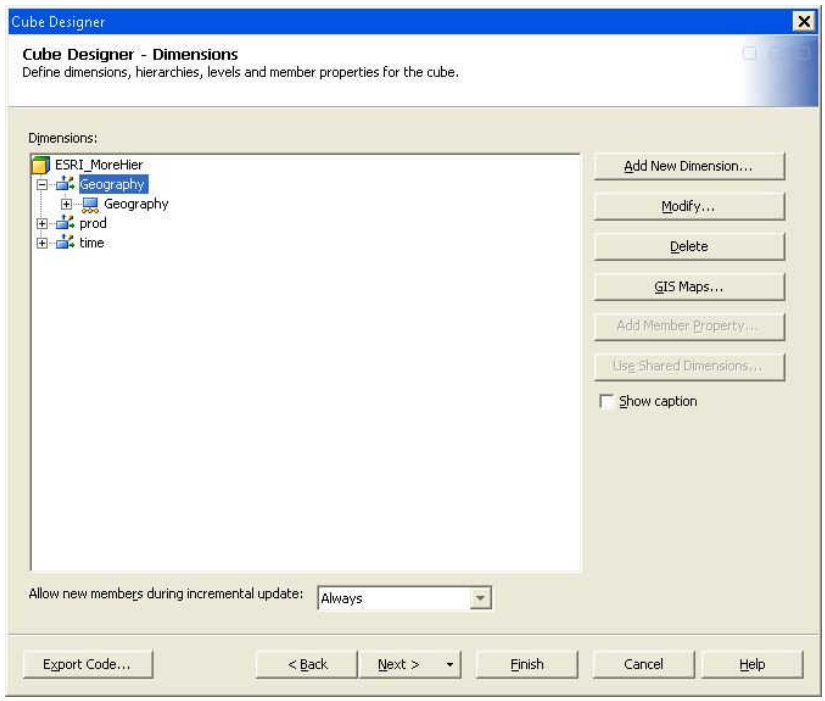
Specify any level properties on the Dimension Designer – Levels page.



Create the hierarchy for the dimension on the Cube Designer – Hierarchy page. Click **Add** to select the dimension levels for the hierarchy or click **Finish** to automatically generate a hierarchy that contains all selected levels for the dimension.

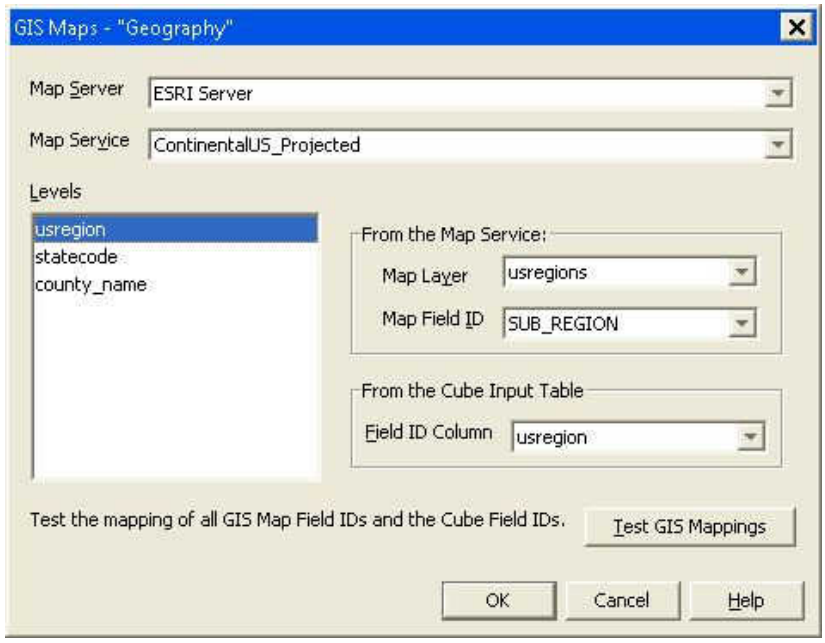


After the dimension is created, it will appear in the **Dimensions** list on the Cube Designer – Dimensions page. In addition, the **GIS Maps** button will be activated.



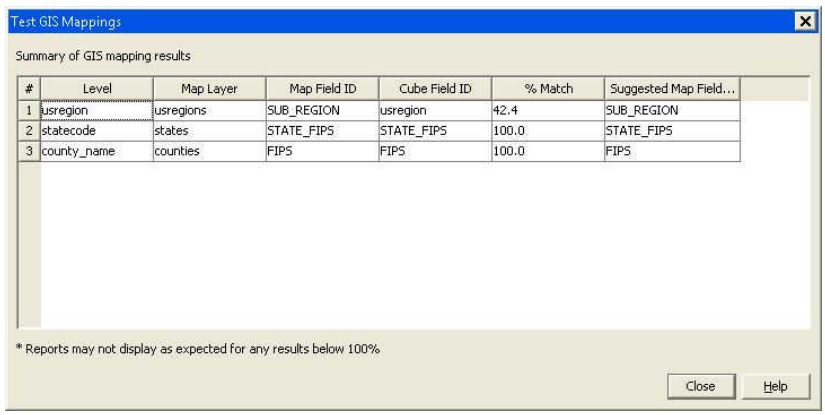
Click **GIS Maps** to open the GIS Maps dialog box. This dialog box enables you to assign Esri spatial map information to levels of the GEO-type dimension. You can now specify the connection settings for the Esri map service.

Select from the drop-down lists the GIS **Map Server** and **Map Service** that you need to connect to. You can now map individual levels to the map service. Select a level from the **Levels** list. Select the **Map Layer** and **Map Field ID** of the map service. Then select the corresponding **Field ID Column** for the cube input table.



You can also test the mapping of the Map Service IDs and the Cube Input Table ID. The following message is displayed. Select **OK** to continue with the testing.

The Test GIS Mappings dialog box is displayed when the testing is complete. Select **Close** to return to the GIS Maps dialog box.



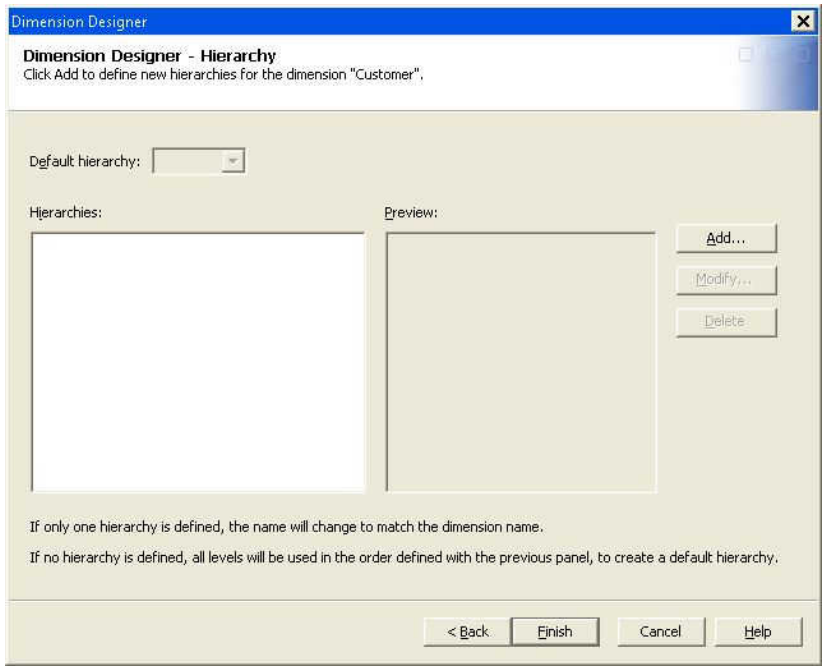
When you are finished, click **OK** on the GIS Maps dialog box to return to the Cube Designer – Dimensions page. Finish creating or editing the cube. Afterwards, you can access the GIS Map functionality through the SAS Web OLAP Viewer.

For further information see “Specifying GIS Map Information for a Dimension” on page 68 .

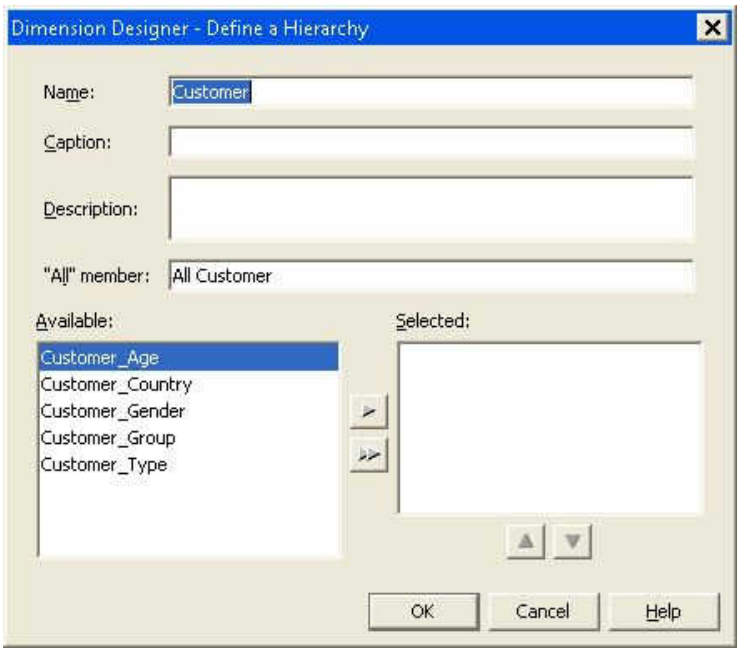
Creating Multiple Hierarchies for a Cube

When you create a cube using the Cube Designer wizard in SAS OLAP Cube Studio, you can specify more than one hierarchy for a dimension. Hierarchies can be defined on the Dimension Designer wizard that is part of the Cube Designer wizard.

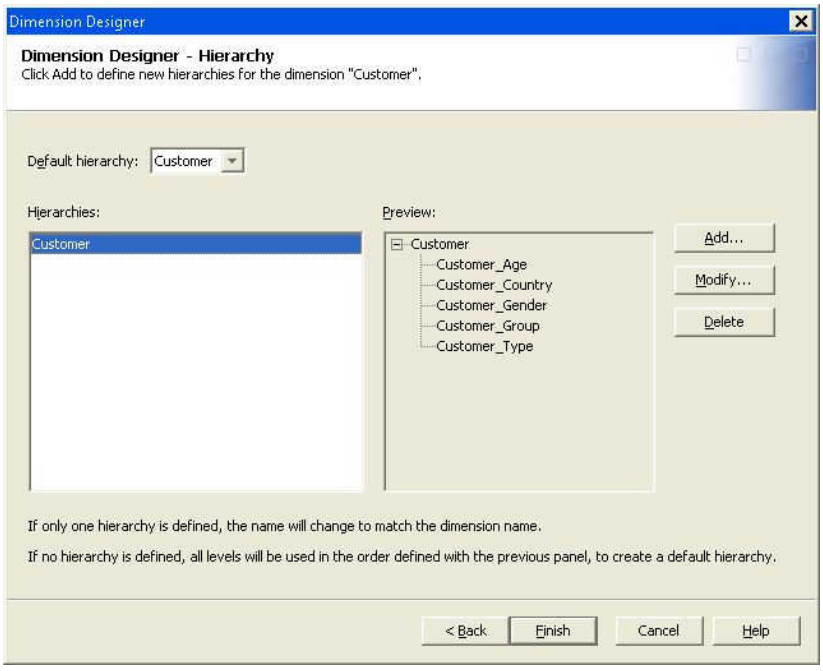
1. Create a cube in the Cube Designer wizard, selecting the input data for the cube and creating dimensions for the cube.
2. On the Dimension Designer – Hierarchy page, click **Add**. See the following display.



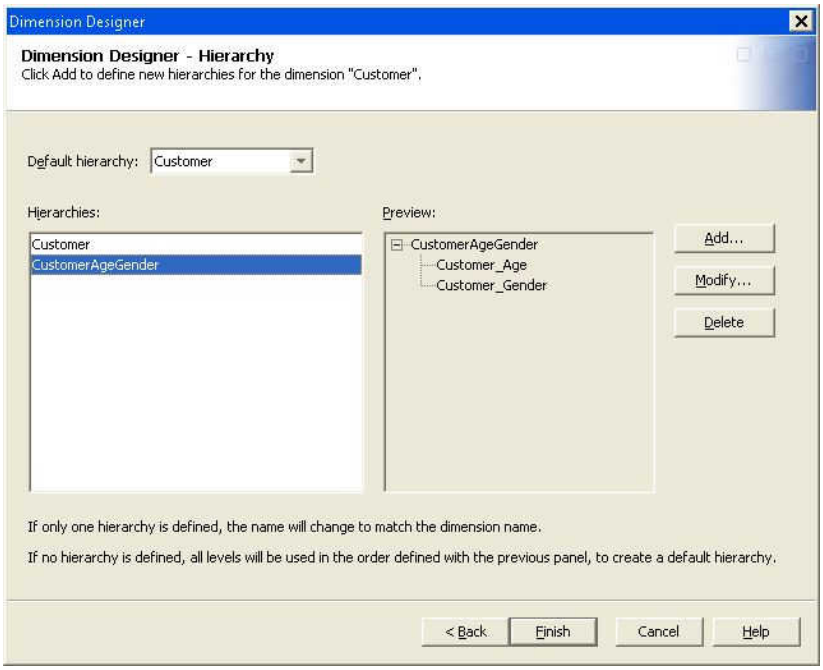
This opens the Dimension Designer – Define a Hierarchy page. See the following display.



3 On the Dimension Designer – Define a Hierarchy page, a hierarchy name is created for you by default that is based on the dimension name. Enter a different hierarchy name if needed. Enter a caption and a description if needed. You can then select the levels for the hierarchy from the **Available** list and add them to the **Selected** list. Select **OK** when you are finished. The hierarchy is created and listed on the Dimension Designer - Hierarchy page. See the following display.



4 Repeat the Dimension Designer – Define a Hierarchy function for each hierarchy that you need to create. As you create hierarchies, they are listed in the **Hierarchies** panel of the Dimension Designer – Hierarchy page. You can then select a default hierarchy from the list of hierarchies that you have created. See the following display:



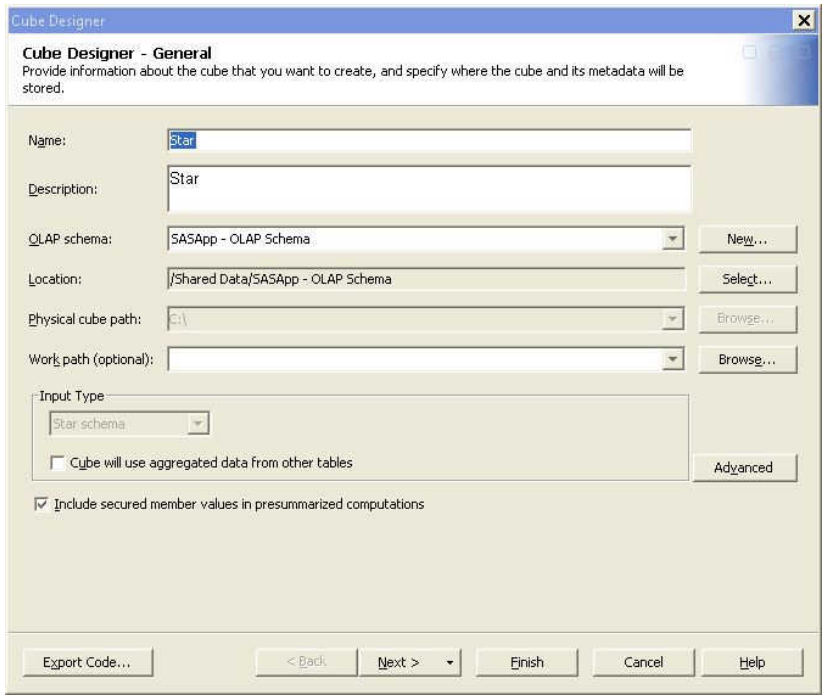
Note: If you are creating a time-specific dimension, use the Add Supplied dialog box to create the levels and hierarchies for the dimension. See the example "Creating a TIME Dimension in SAS OLAP Cube Studio" on page 258.

If you are creating the cube using PROC OLAP, you will include a HIERARCHY statement for each hierarchy that you create for the dimension. See the following example PROC OLAP code:

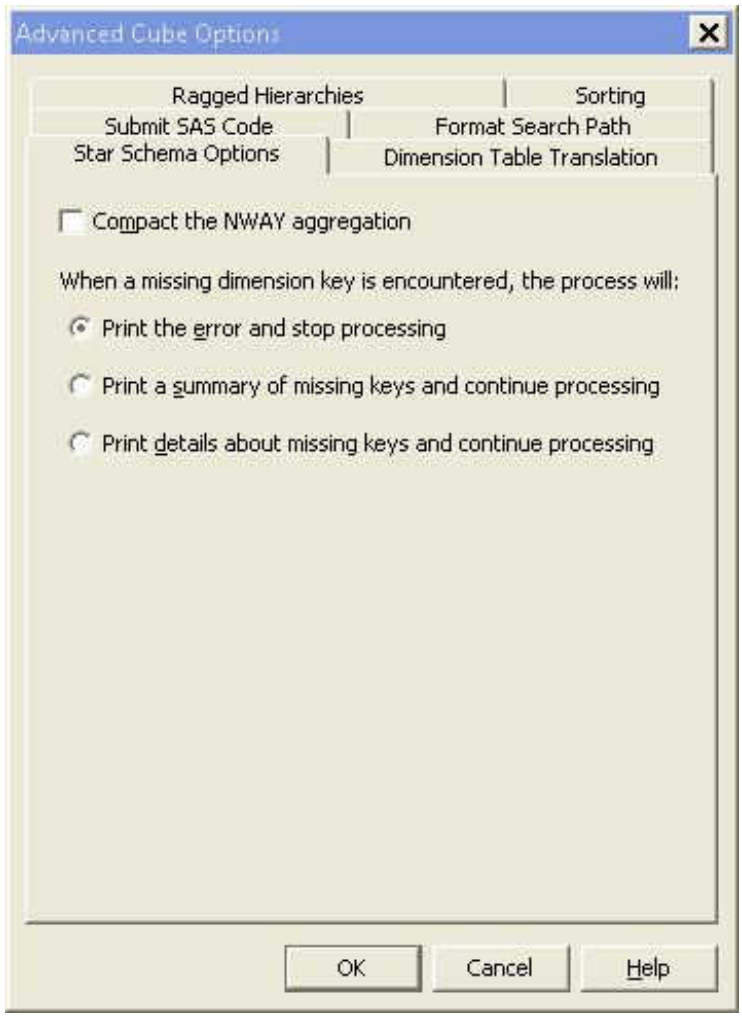
```
DIMENSION Customer
   CAPTION          = 'Customer'
   HIERARCHIES      = (
      Customer CustomerAgeGender
      ) /* HIERARCHIES */;

   HIERARCHY Customer
      ALL_MEMBER = 'All Customer'
      CAPTION    = 'Customer'
      LEVELS     = (
         Customer_Age Customer_Country Customer_Gender
         Customer_Group Customer_Type
         ) /* LEVELS */
      DEFAULT;

   HIERARCHY CustomerAgeGender
      ALL_MEMBER = 'All CustomerAgeGender'
      CAPTION    = 'CustomerAgeGender'
      LEVELS     = (
         Customer_Age Customer_Gender
         ) /* LEVELS */;
```

Set IGNORE_MISSING_DIMKEYS for a Star Schema

If you are building a cube with a star schema data source, you can define how the cube build process will respond to missing dimension keys. A missing dimension key is detected when the star schema fact table contains foreign key values that are not present in one of the corresponding dimension tables. You can set the option in SAS OLAP Cube Studio when you are creating or editing a star schema cube. On the Cube Designer – General page, click **Advanced**.



This opens the Advanced Cube Options dialog box. Select the **Star Schema Options** tab. You can now select one of the options for handling a missing dimension key if it is encountered.



Here are the options:

Print error and stop processing
Any missing dimension keys stop the build of the cube. This is the default option.

Print summary of missing keys and continue processing
The cube build continues and the fact table row with the missing key is ignored (it is not built into the cube).

Print details about missing keys and continue processing
The cube build continues and the fact table row with the missing key is ignored. In addition, the log receives additional detail and the missing keys are listed for each dimension table.

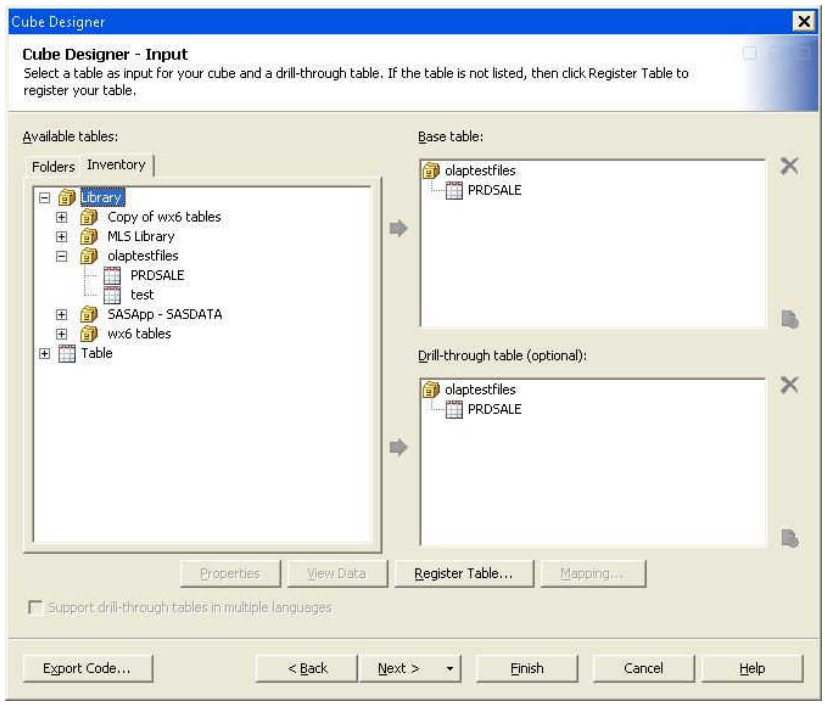
You can also set the missing key option when you are updating a cube. See the example “Generating a New Cube” on page 227 . If you are defining a cube in PROC OLAP you can set the IGNORE_MISSING_DIMKEYS option in the PROC OLAP statement.

Implementing Drill-through to Detail Data in a SAS OLAP Cube

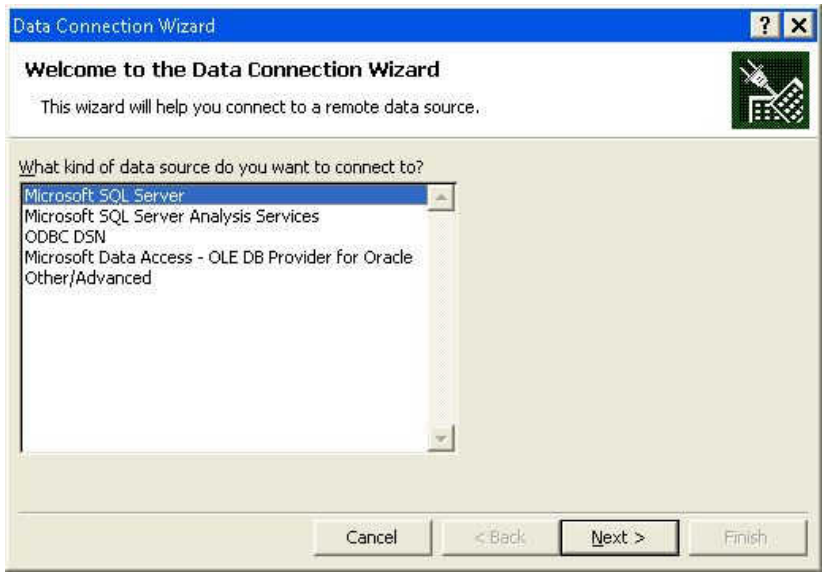
Many SAS OLAP applications give users the ability to select a cell or a range of cells and then view the input data that the cell data was summarized from. SAS OLAP enables you to assign a drill-through table to a cube. You can then access the underlying cell data.

Note: When you select a data table for drill-through, you might need to consider user access and security restrictions for that table. For further information see “Security for Drill-through Tables” on page 126 .

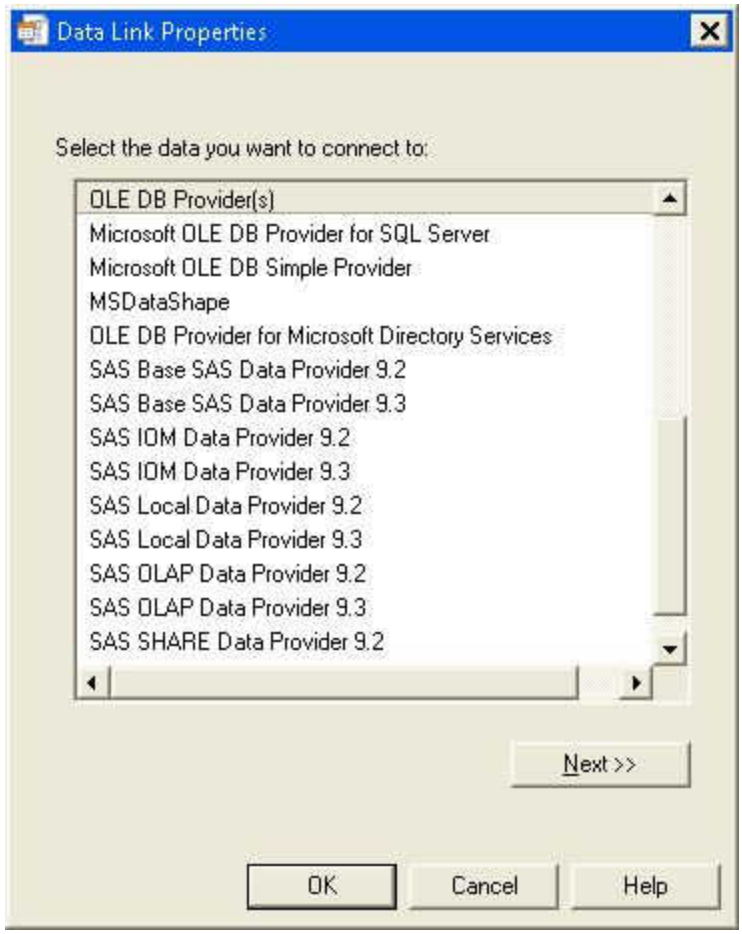
In this example, a cube is assigned a drill-through table, and then the cube is accessed in Microsoft Excel 2007. In SAS OLAP Cube Studio, open a cube in the Cube Designer wizard. In the Cube Designer – Input page, assign a drill-through table. This is often the same table that is used as the input table for the cube. Click **Finish** to save the drill-through table assignment to the cube. See the following display.



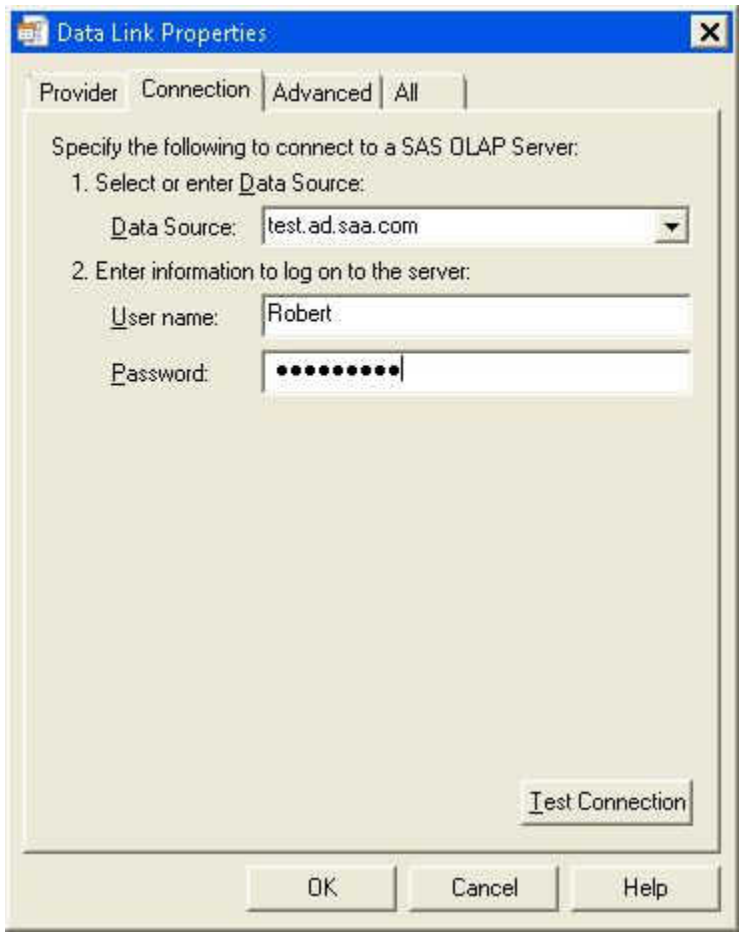
In Microsoft Excel 2007, select **Data** ⇨ **From Other sources** ⇨ **From Data Connection Wizard**. This opens the Data Connection Wizard. Select the option **Other/Advanced**. Select **Next**.



On the Data Link Properties page, select the **Provider** tab. In the list of data provider options, select the appropriate release of the **SAS OLAP Data Provider**. Click **OK** to return to the Data Link Properties dialog box.

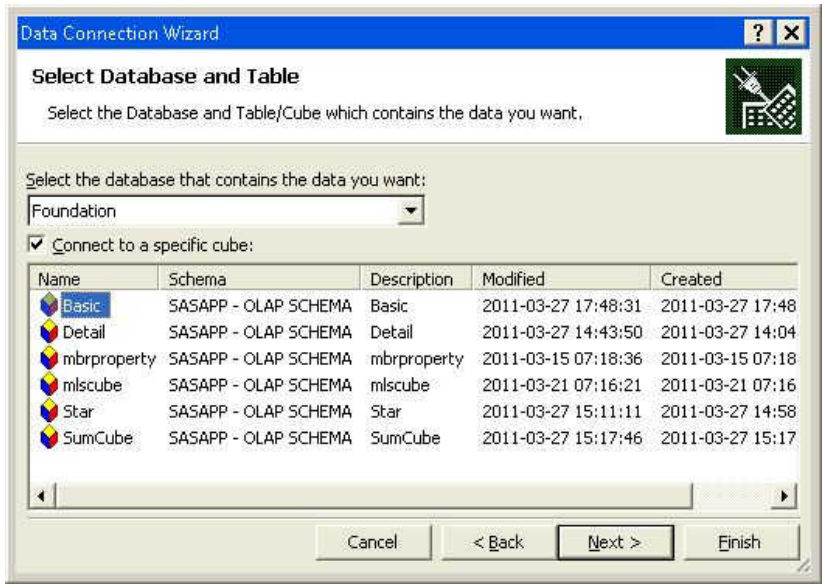


On the **Connection** tab, enter the **Data Source**, **User name** and **Password** for your SAS OLAP Server.

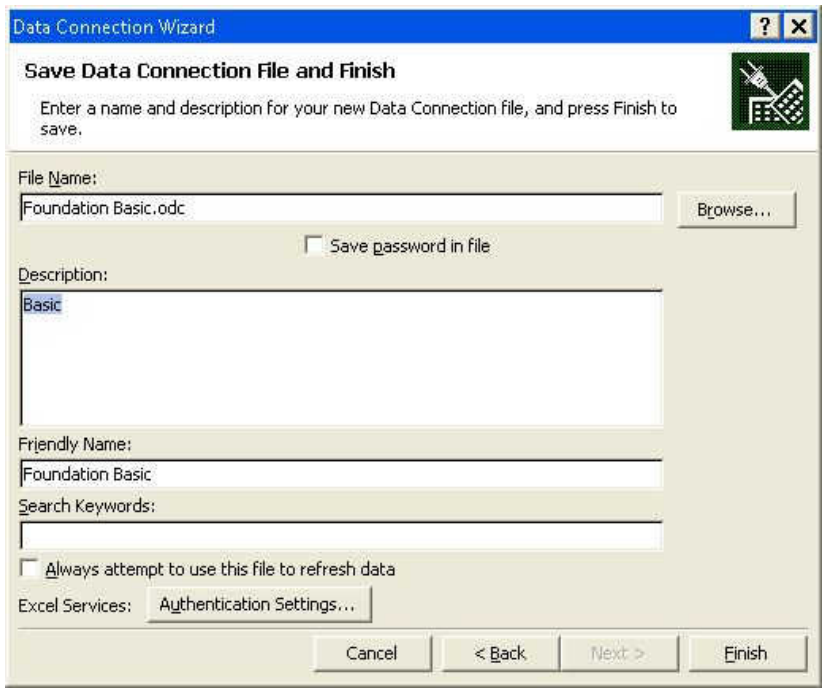


Click **Test Connection** to test the connection to the SAS OLAP Server. Click **OK** to respond to the **Connection Succeeded** message and return to the Data Link Properties dialog box. Click **OK** to return to the Data Connection Wizard.

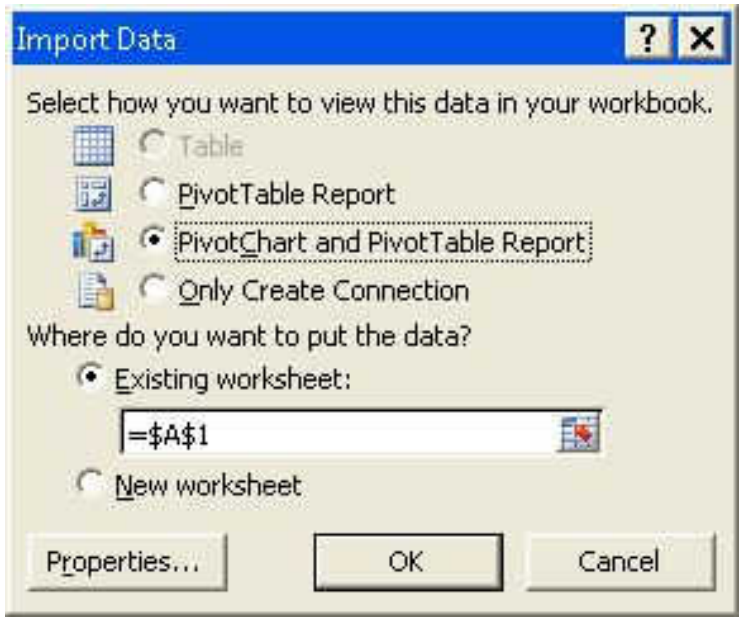
On the Select Database and Table page, select a database and a cube, and then click **Next**.



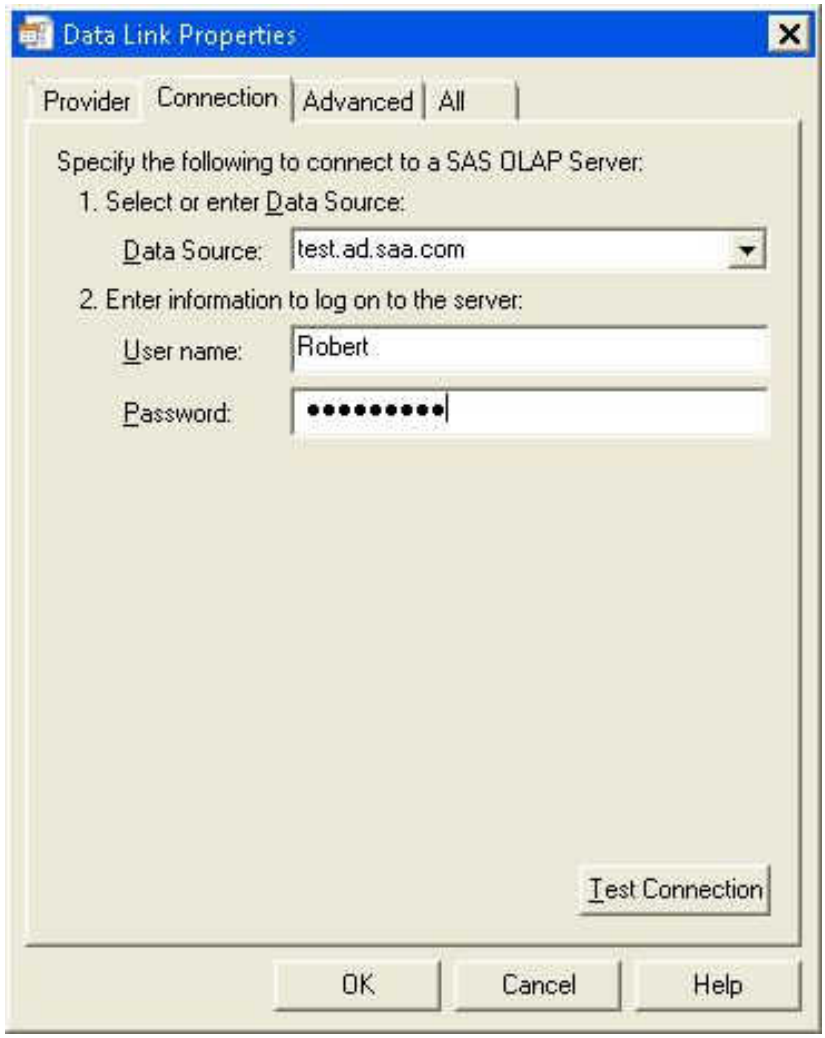
On the Save Data Connection File and Finish page, enter data that will help you and others access the cube and understand its purpose. You can enter a **Description**, **Friendly Name**, and **Search Keywords** for the cube. Click **Finish** to complete the Data Connection Wizard.



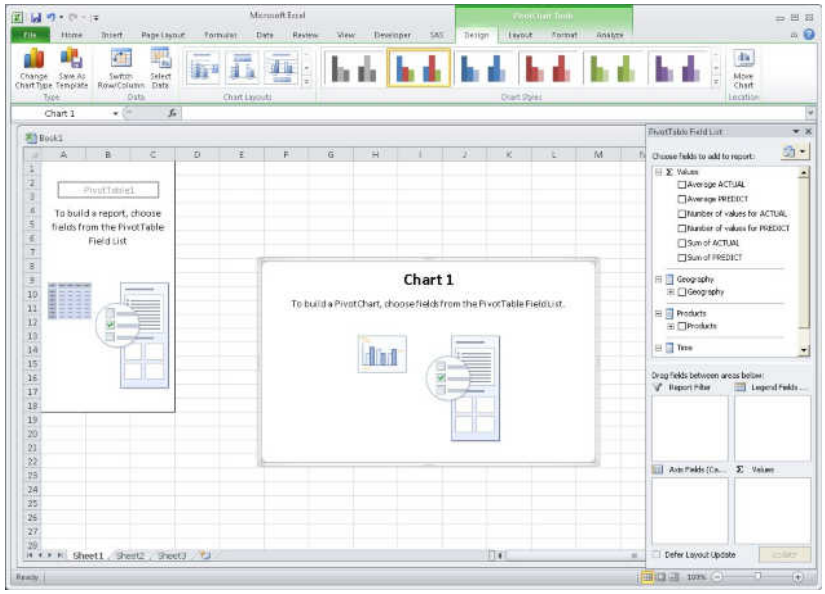
The Import Data dialog box appears. Select to view the cube data in a **PivotTable Report** or in a **PivotChart and a PivotTable Report**. Click **OK**.



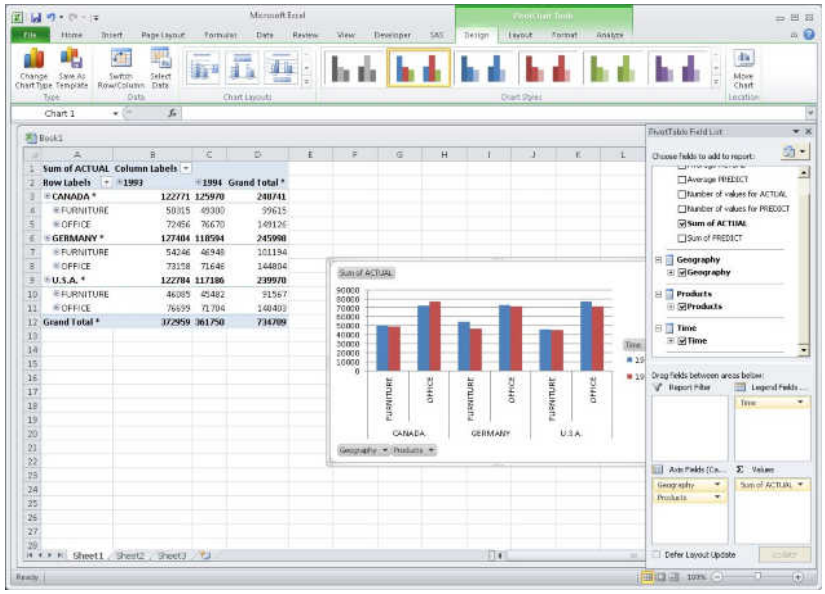
The Data Link Properties dialog box appears. Enter your password and click **OK**.



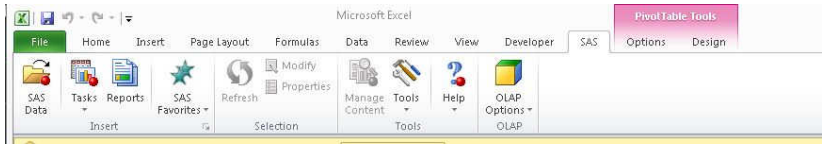
The pivot table opens, along with the pivot chart, if it is selected. From here you can select fields to add to your pivot table and pivot chart.



As you select fields to add, the pivot table and pivot chart become populated. You can now view the drill-through data for a particular cell in the pivot table. Select the cell that you want to drill down to.

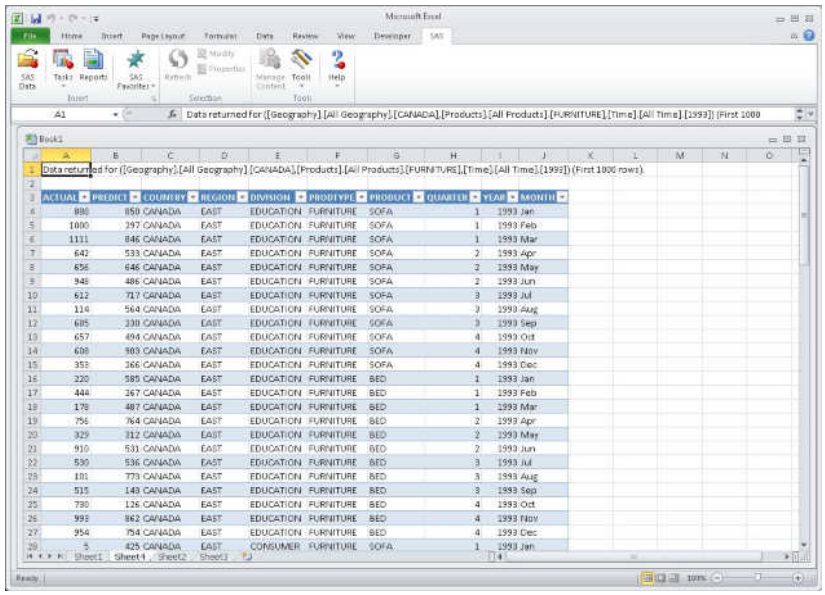


On the **SAS** menu, select the **OLAP Options** menu.



On the **OLAP Options** menu, select the **Drill through Details** option.

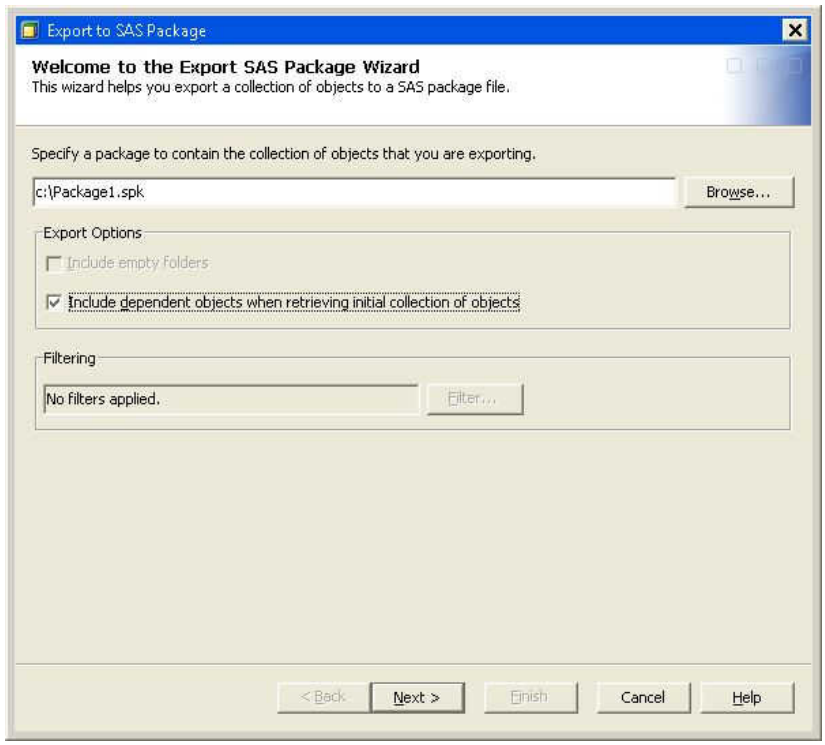
The detail data for the cell in the pivot table is displayed on a second sheet in Excel.



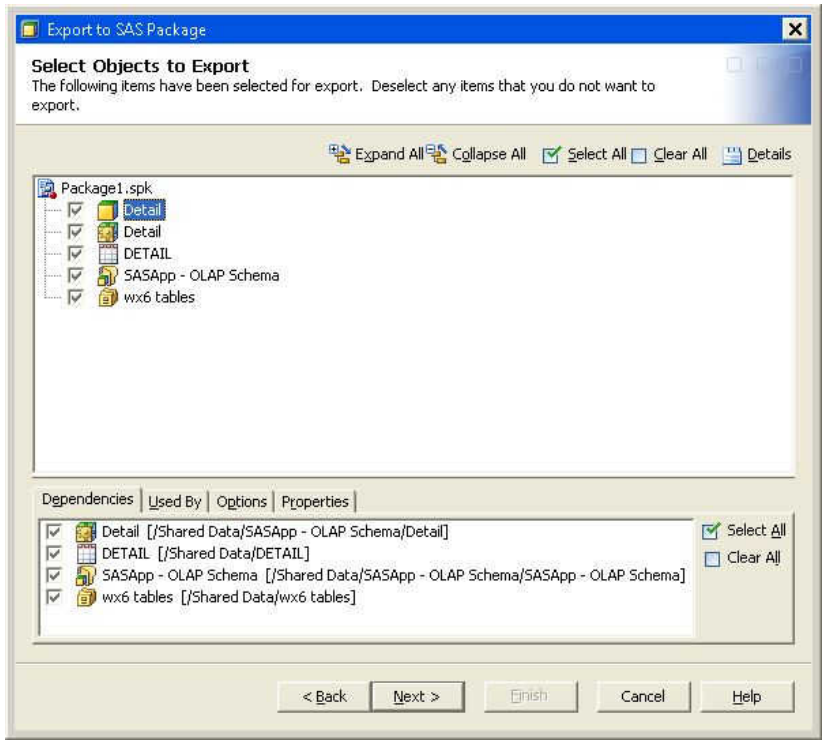
Exporting a Cube from SAS OLAP Cube Studio

In SAS OLAP Cube Studio, you can select a cube to export as part of a SAS package. Select a cube in the tree view. Select **File** ⇨ **Export SAS Package**. The Export SAS Package wizard opens.

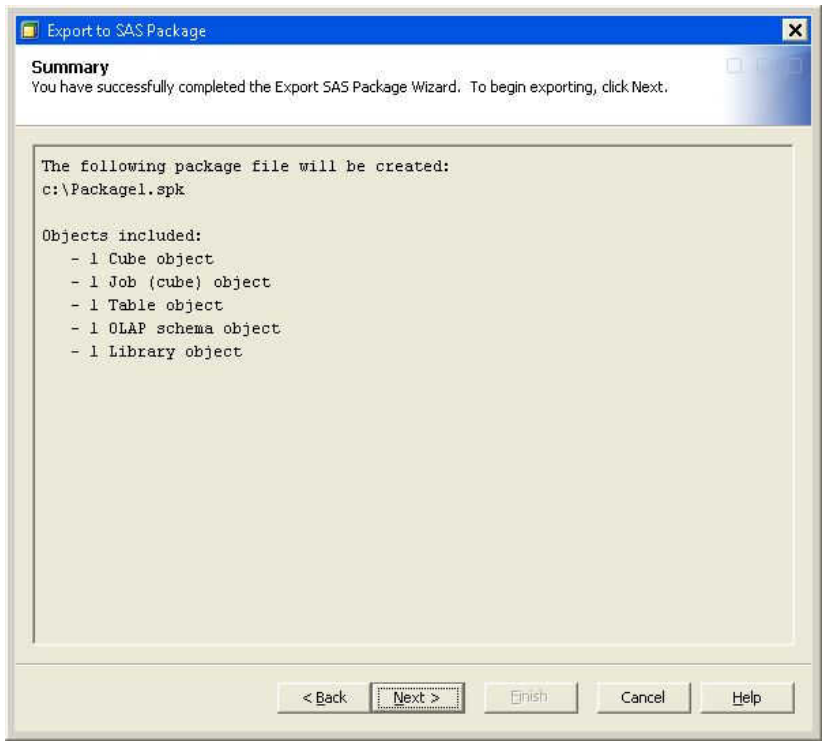
On the Welcome to the Export SAS Package Wizard page, enter the name of the package that you want to export. A default package named **Package1.spk** is automatically entered. You can also check the option **Include dependent objects when retrieving initial collection of objects**. This enables you to automatically include all dependent objects, recursively, in the export. Select **Next**.



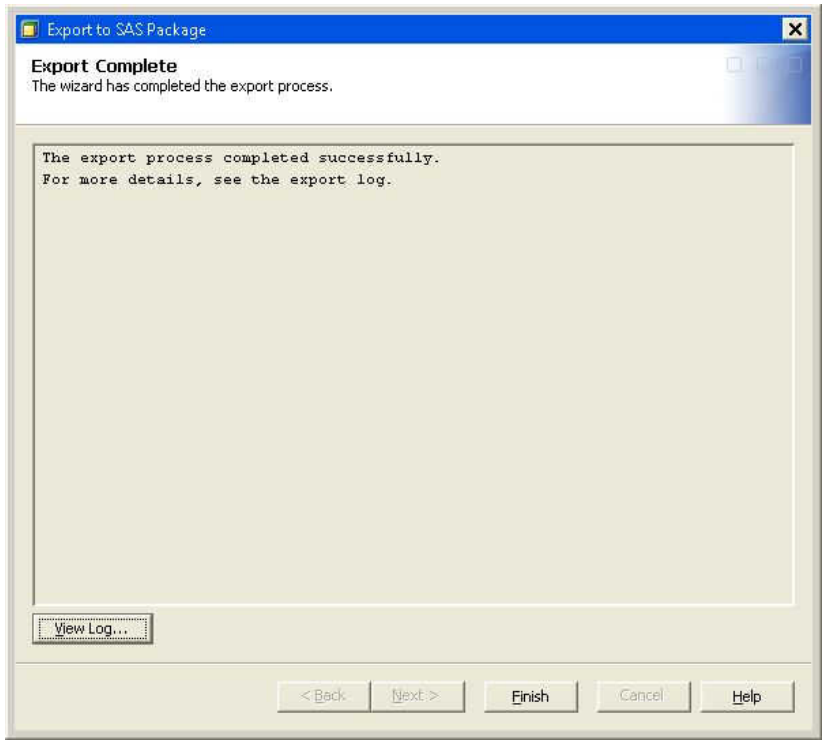
On the Select Objects to Export page, check the objects that you want to include in the exported package. The cube job is automatically selected with the cube. Click **Next**.



On the Summary page, review the objects that are included in the exported package. Select **Next**.



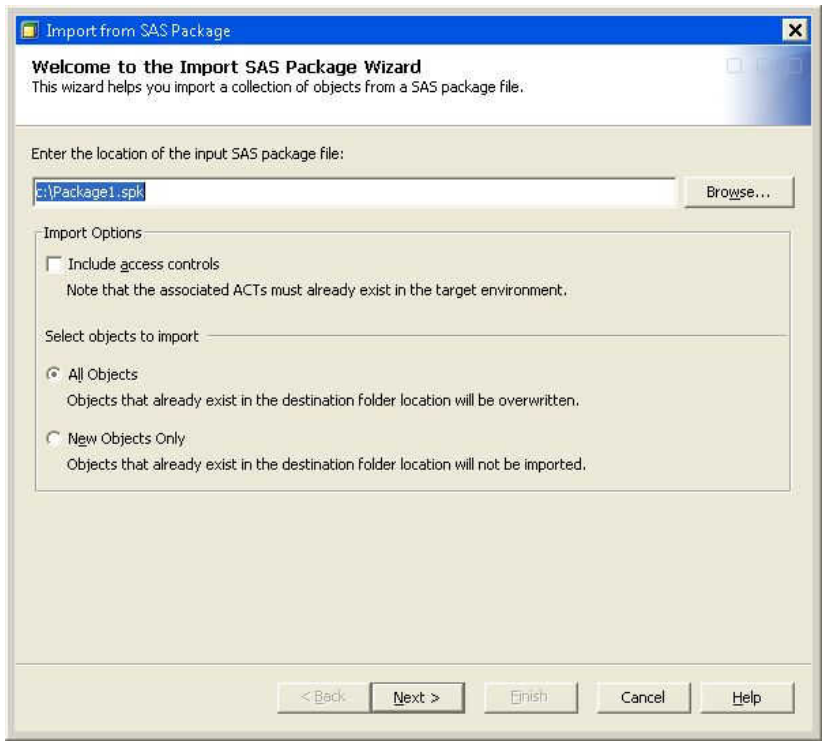
On the Export Complete page, you can view the status of the export process. You can also click **View Log** to view the Export Log. Select **Finish** to complete the Export SAS Package wizard.



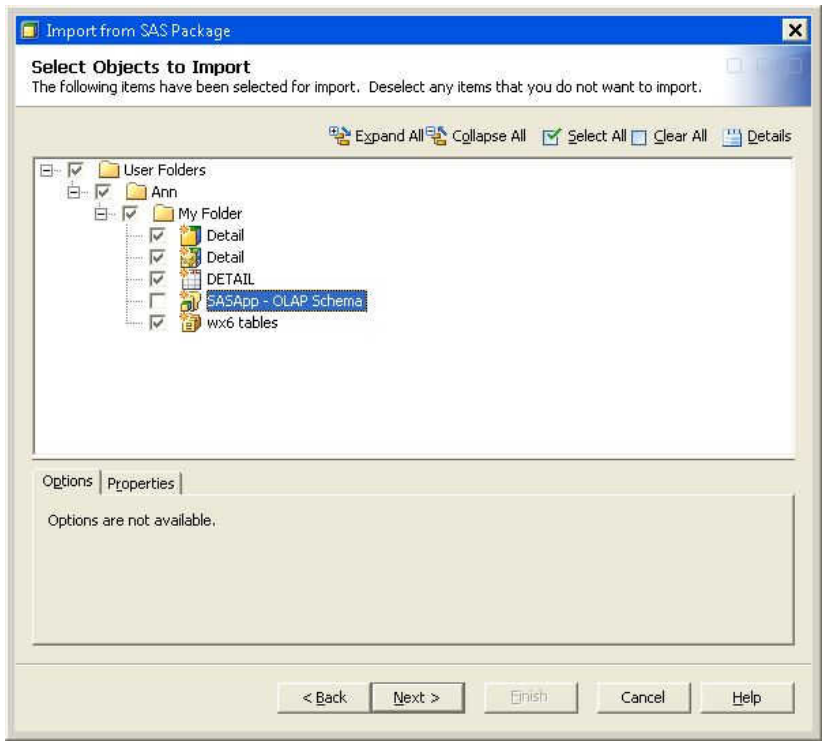
For detailed information about exporting SAS OLAP cubes, see “Export SAS Package and Import SAS Package” on page 306 .

Importing a Cube into SAS OLAP Cube Studio

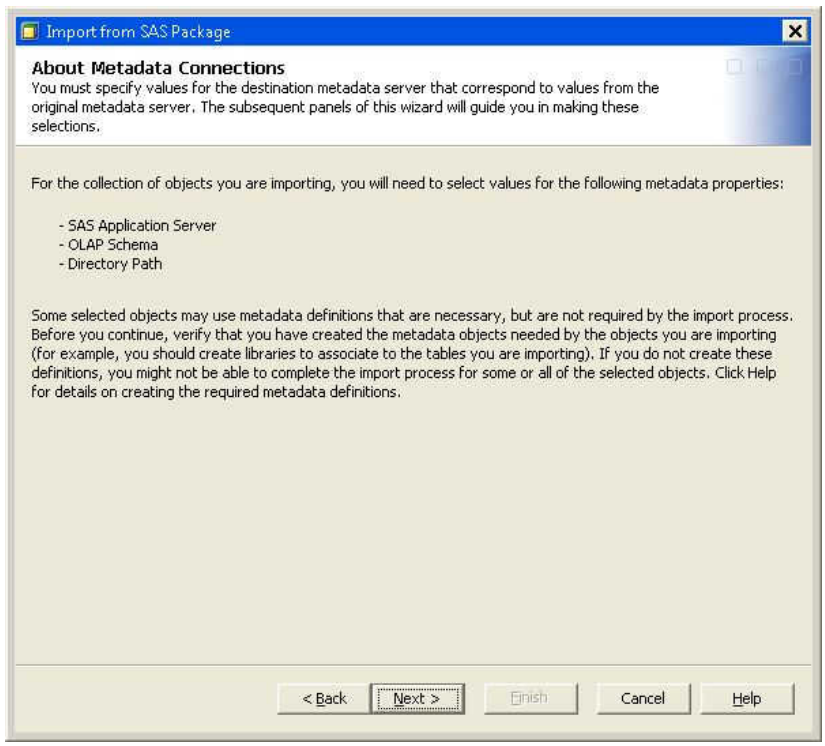
In SAS OLAP Cube Studio, you can import a cube that has been previously exported as part of a SAS package. In the tree view, select the **Folders** tab. Select a folder to import a cube to. Select **File** ⇨ **Import SAS Package**. The Import SAS Package Wizard appears. On the Welcome to the Import SAS Package Wizard page, select the package that you want to import. On the **Import Options** panel, select the import options that you need. Select **Next**.



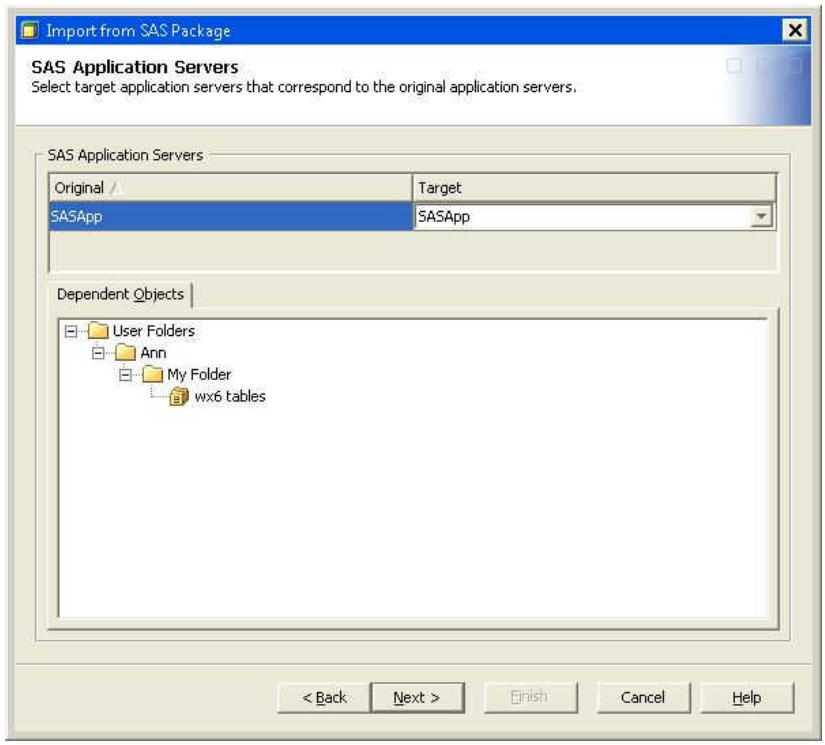
On the Select Objects to Import page, select the objects from the SAS package that you want to import. Select **Next**.



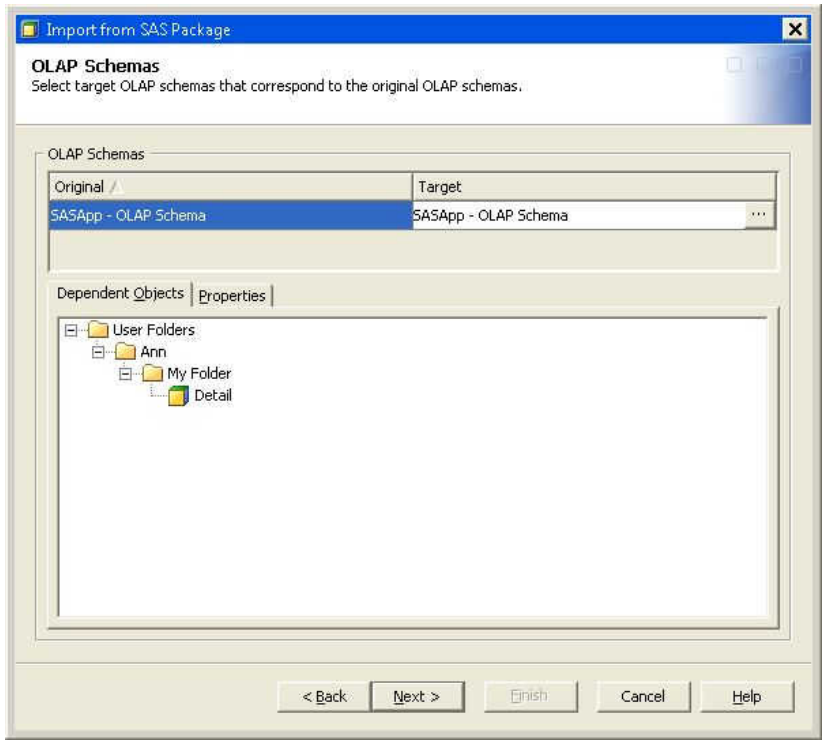
The About Metadata Connections page lists the metadata properties that you need to select values for. This list is determined from the objects that you selected to import. In this example, a SAS Application Server and a directory path must be selected in the following pages of the wizard. Select **Next**.



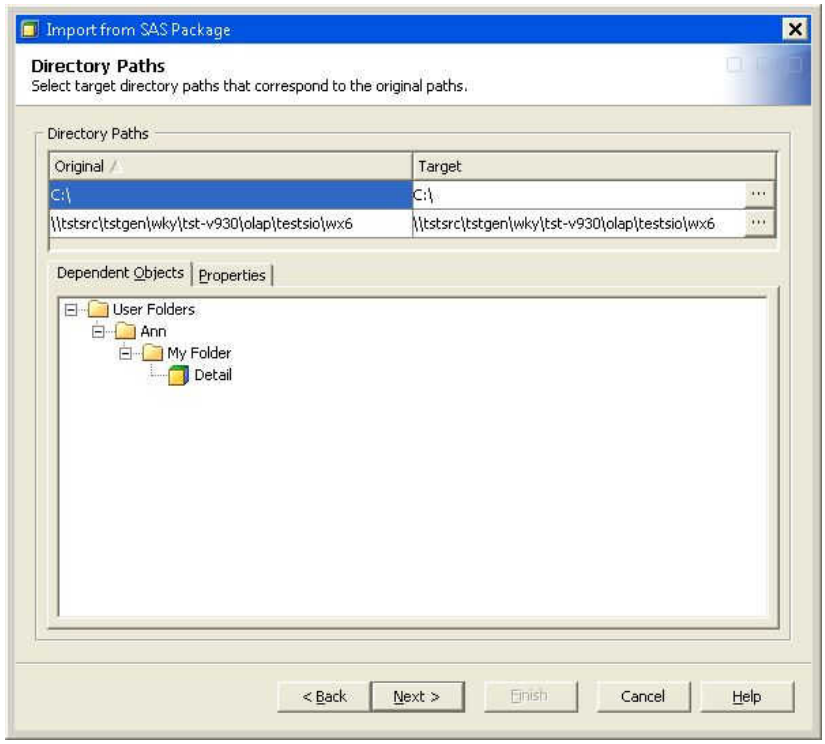
On the SAS Application Servers page, select the target SAS Application Server. Select **Next**.



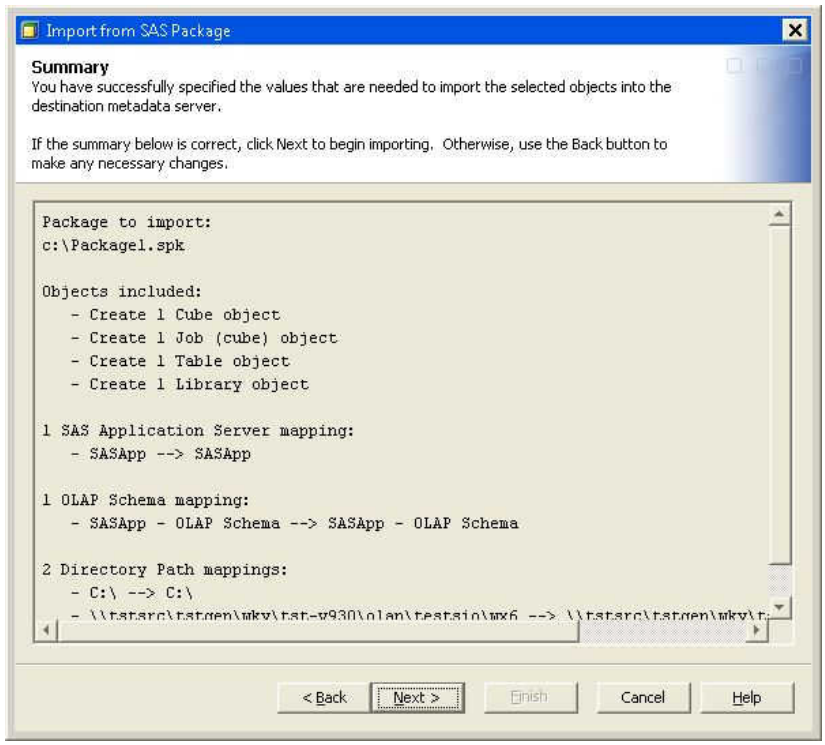
On the OLAP Schemas page, select the OLAP schema on the target system that you are assigning the imported cube to. You must select an OLAP schema when importing SAS OLAP cubes. Select **Next**.



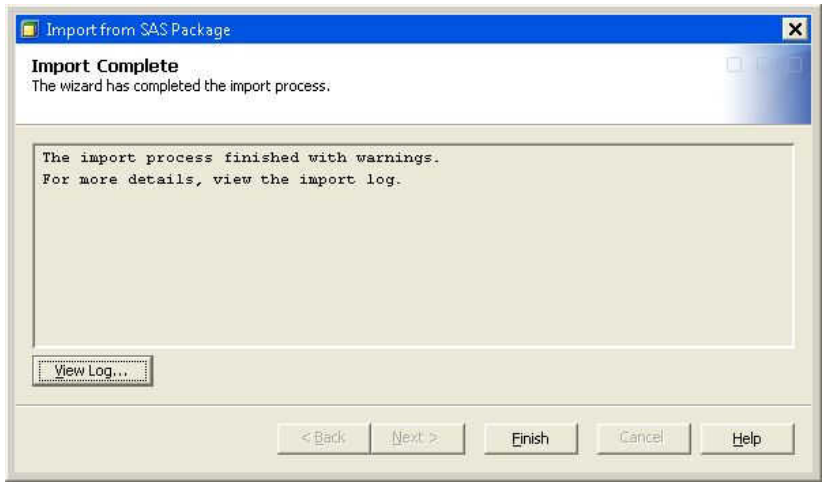
On the Directory Paths page, select the directory path, on the target system, that you want to import the cube files to. Select **Next**.



On the Summary page, review the objects that you want to import. Select **Next**.



On the Import Complete page, you can view the status of the import process. You can also click **View Log** to view the Import Log. Select **Finish** to complete the Import SAS Package wizard.



For detailed information about importing SAS OLAP cubes, see “Export SAS Package and Import SAS Package” on page 306 .

Building a Shared Dimension

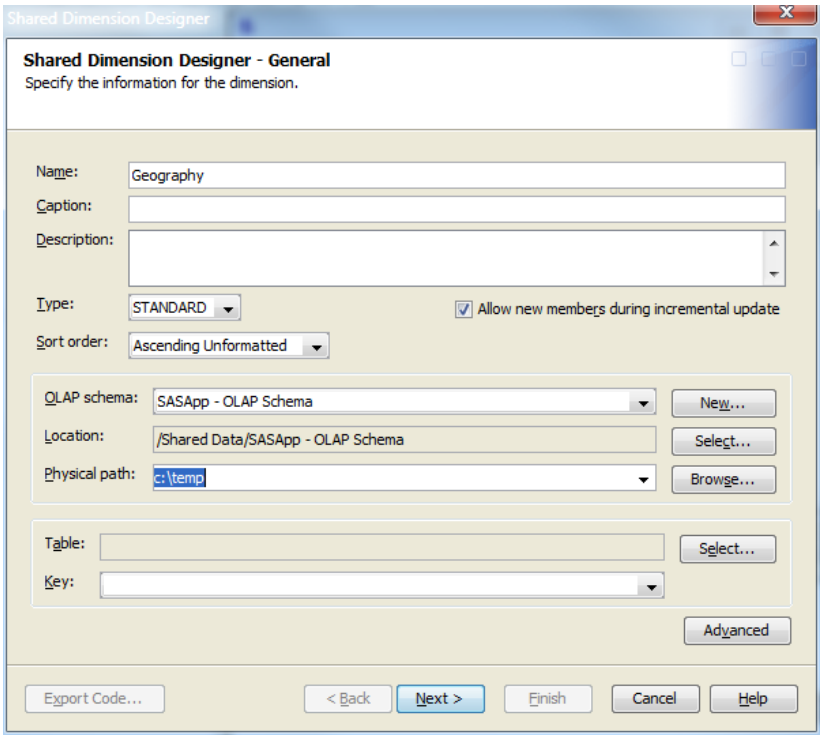
Introduction

When you work with SAS OLAP cubes, it is often the case that you share dimensions across many cubes. Updates to the dimension can occur over time. The cubes that contain that dimension must be able to reflect those changes. A SAS shared dimension provides a common dimension that is created and updated in one place and is automatically reflected across all cubes that use the dimension.

Shared dimensions are built from a single dimension table. You can build a shared dimension using the Shared Dimension Designer in SAS OLAP Cube Studio. In this example, you build a geography dimension for a product marketing campaign and then use the shared dimension in a cube.

Enter General Information for the Shared Dimension

After you have established a connection profile on page 153, you can begin to create the shared dimension. Select **File** ⇨ **New** ⇨ **Shared Dimension**. On the Shared Dimension Designer – General page, enter the basic dimension information, as shown in the following display.



Enter data in the following fields:

- **Name**
- **Caption**
- **Description**
- **Type** (GEO, STANDARD, TIME)
- **Sort order**
- **OLAP schema**
- **Location** (SAS folder)
- **Physical path**

Select a Dimension Table

Click the **Select** button to the right of the **Table** field to select a shared dimension table. The Select dialog box appears. If the table does not exist for your dimension, click **Register Table** and then define the source that you will import your metadata from. The following display shows the selection of the dimension table GEOGDIM.

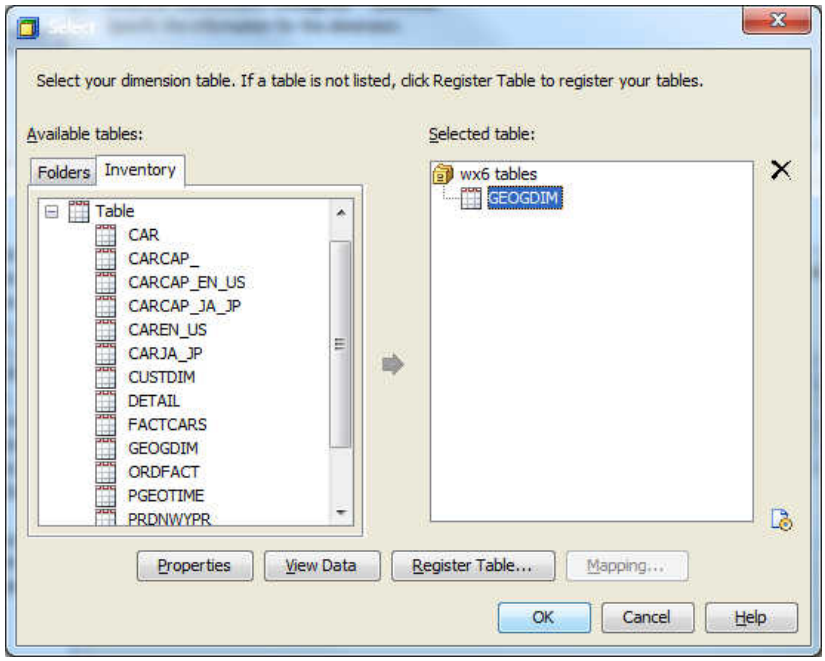


Table Options

To specify options that are used to open the table, click the button next to the **Selected table** box. In the Table Options dialog box, specify data set options that are used to open the data set. For example, you could enter a WHERE clause or other information to subset or filter the data in the table as it is opened.

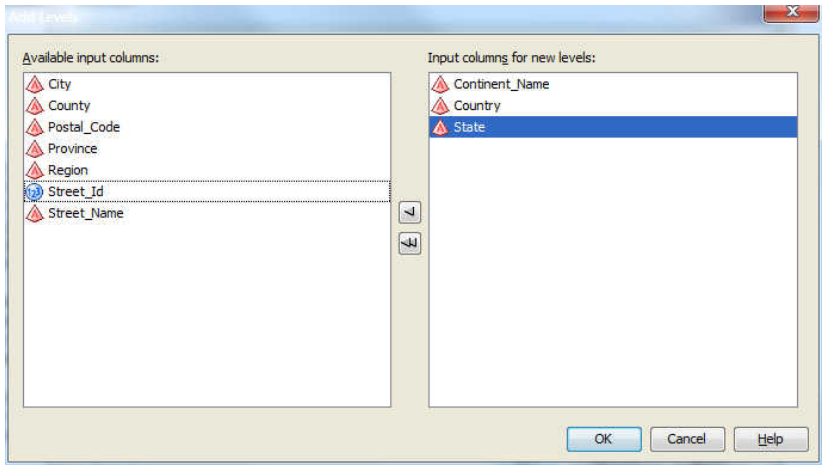
The table options are stored as part of the cube and then reapplied when the data is accessed at run time. You can also specify data set options in the Dimension Designer – General page (for use with star schemas) and the Stored Aggregates dialog box (for use with summarized tables). For more information, see “Data Set Options” in *SAS Language Reference: Concepts*.

Select a Dimension Key

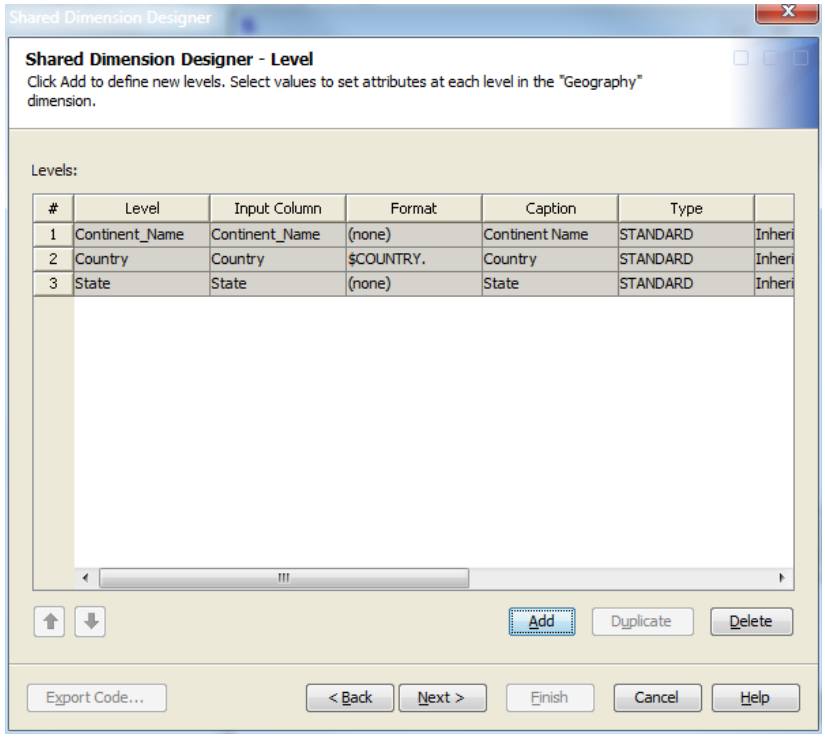
After you select a dimension table, you return to the Shared Dimensions – General page to specify a key column for that dimension. The key column in the shared dimension table links the records in that table to the records in the fact table of the star schema. Use the menu in the **Key** field to select the key column.

Note: A key column is not needed for the shared dimension definition, but it is required when you associate a shared dimension to a cube.

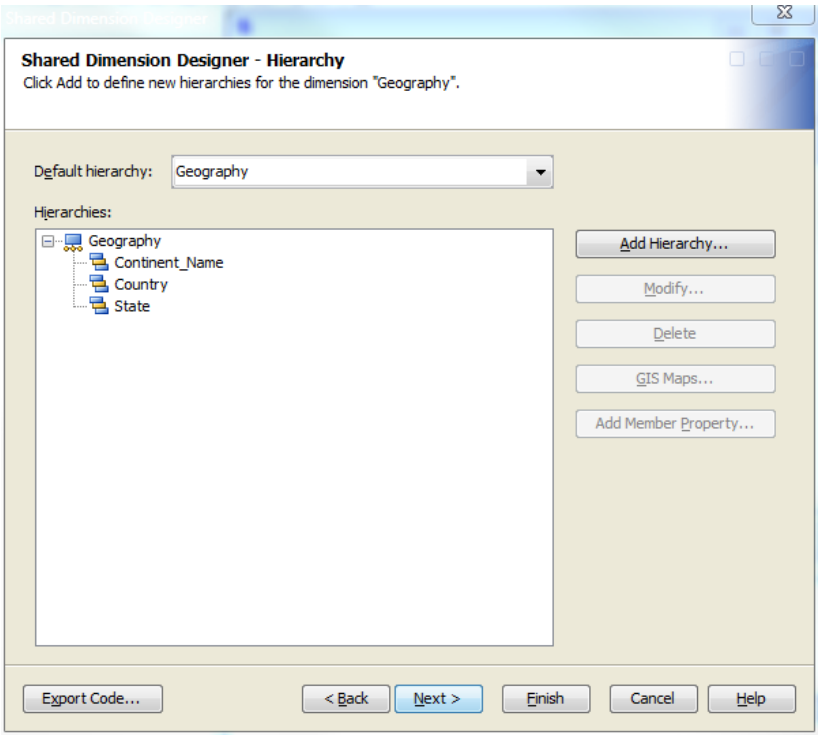
Click **Next** to open the Dimension Designer – Levels page, and then click **Add** to open the Add Levels dialog box.



In the Add Levels dialog box, select the levels that you want to add to the dimension, and then click **OK** to return to the Shared Dimension Designer – Level page. You can now define properties such as format, time type, and sort order for the levels that you have selected. See the following display.



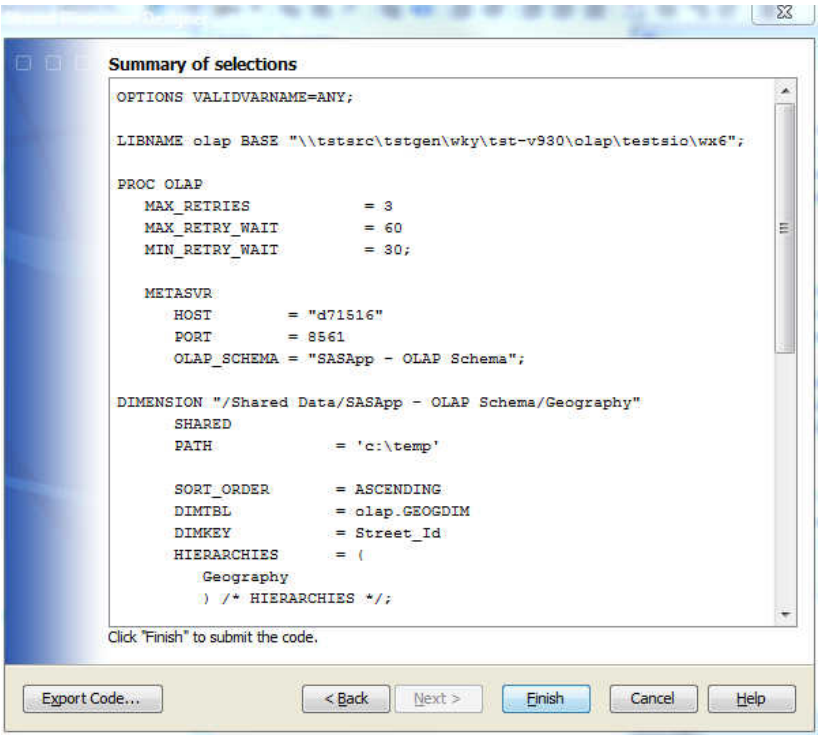
When your levels are complete, click **Next** to display the Shared Dimension Designer – Define a Hierarchy page. In that page, you apply levels to hierarchies. A default hierarchy is created for you. The default hierarchy includes all levels, in the order in which they appear in the previous Level page, as shown in the following display.



Click **Add Hierarchy** to add other hierarchies to the dimension, and then click **Next** to display the Summary of Selections page.

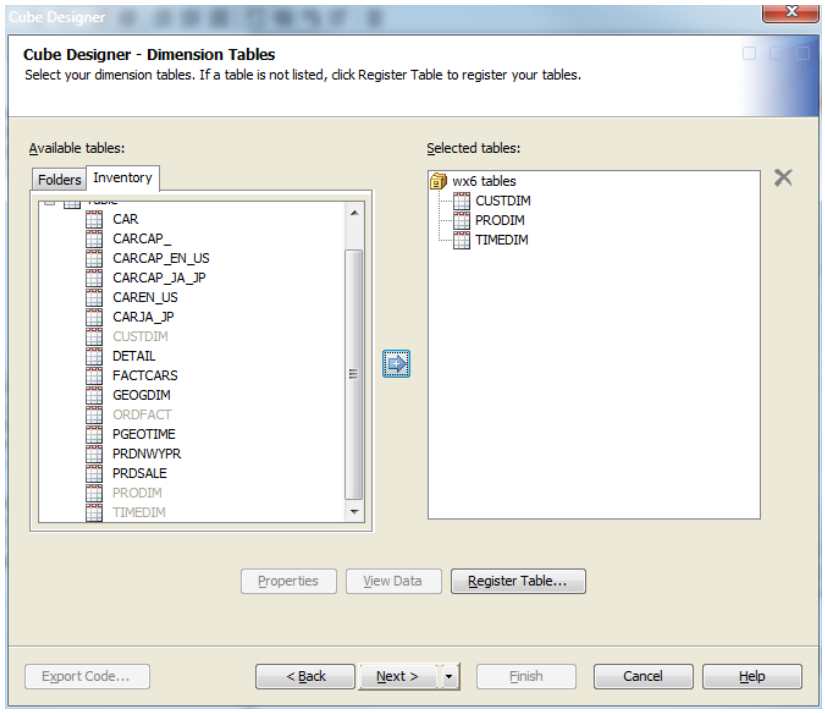
Build the Shared Dimension

You can now build the shared dimension. On the Summary of Selections page, review the specifications for your shared dimension, export your PROC OLAP code, and click **Finish** to build the shared dimension.

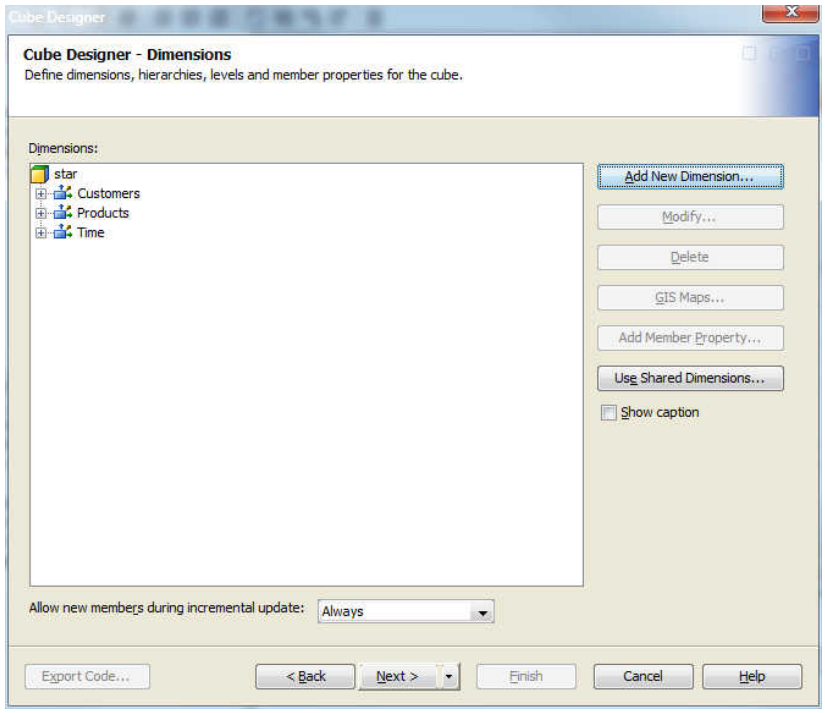


Using a Shared Dimension in a Cube

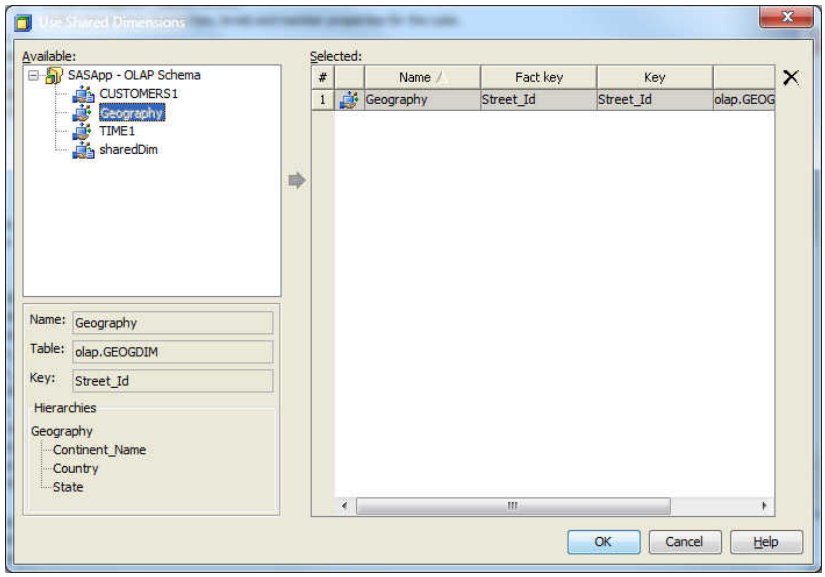
Shared dimensions are used in cubes that are built from star schemas. To use a shared dimension, open the Cube Designer and begin defining the cube, as described in “Building a Cube from a Star Schema ” on page 175. When you arrive at the Cube Designer – Dimension Tables page, do not select the table for the shared dimension. Select all of the other tables, but do not select the shared dimension table. The following display shows that the shared dimension table GEOGDIM has not been selected.



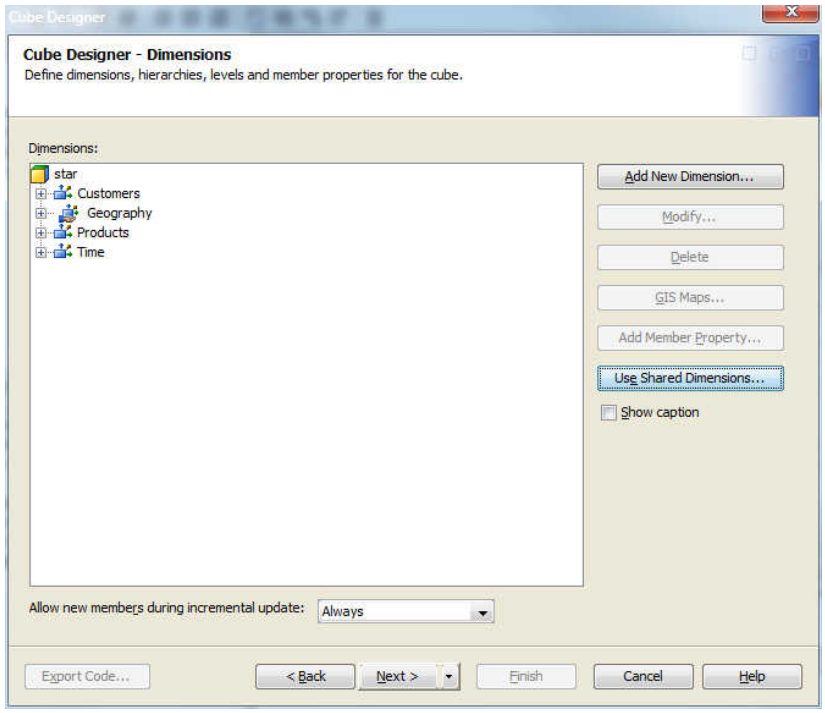
When you complete your selection of non-shared dimension tables, click **Next** to return to the Cube Designer – Dimensions page.



When you have configured your non-shared dimensions, click **Use Shared Dimensions** to display the Use Shared Dimensions dialog box, as shown in the following display.



Select the shared dimension and specify the key column in the fact table that links to the key column in the shared dimension table. In this example the **Street_ID** column from the fact table ORDFACT links to the **Street_Id** column of the shared dimension table GEOGDIM. Select **OK** to return to the Cube Designer – Dimensions page. The shared dimension is listed with the other dimensions, as shown in the following display.



Continue with the remaining steps to build the cube, by selecting measures and specifying aggregations. The levels and hierarchies of the shared dimension can be used to create measures and aggregations along with the levels and hierarchies of the non-shared dimensions.

10

Using SAS OLAP Cubes

Using a Cube with ADO MD

Applications gain access to SAS OLAP cubes through ActiveX Data Objects – Multidimensional (ADO MD). ADO MD is an industry-standard programming interface to multidimensional data. It offers the same functionality as OLE DB for OLAP, but in a simpler programming model. Accessing SAS OLAP cubes through ADO MD requires the SAS OLAP Data Provider, which is a component of SAS Integration Technologies. The SAS OLAP Data Provider is installed with the SAS Integration Technologies Client for Windows. See the *SAS Providers for OLE DB: Cookbook* for more information about IOM data provider usage with ADO MD.

Using a Cube with OLE DB for OLAP

In addition to ADO MD, applications gain access to SAS OLAP cubes through OLE DB for OLAP (ODBO), an industry-standard set of programmable Component

Object Model (COM) interfaces that expose multidimensional data. For SAS OLAP cubes, the OLE DB interfaces are exposed by the SAS OLAP Data Provider, a component of SAS Integration Technologies. The SAS OLAP Data Provider enables applications to perform data analysis by providing a means to view schema information, submit MDX queries, and retrieve results. The SAS OLAP Data Provider is installed with the SAS Integration Technologies Client for Windows. See the *SAS Providers for OLE DB: Cookbook* for more information about IOM data provider usage.

Using a Cube with Additional SAS Products

Listed below are the SAS products the enable you to access, view, query, and report on SAS OLAP cubes. Specific OLAP functionality within these products is also listed:

SAS Products That Use SAS OLAP Cubes

- SAS Enterprise Guide
- SAS Information Map Studio
- SAS Web Report Studio
- SAS Add-In for Microsoft Office
- SQL Pass-Through Facility for OLAP. For more information, see “Accessing OLAP Cubes from SAS: SQL Pass-Through Facility for OLAP” on page 110 .

  Note: The SQL Pass-Through Facility for OLAP does not require additional licensing.

For information about how to access SAS OLAP cubes in these different products, see the individual product Help and documentation.

SAS Enterprise Guide

SAS Enterprise Guide provides a dedicated, intuitive and advanced interface is provided for analyzing business information stored in OLAP data cubes. SAS Enterprise Guide is an ODBO-compliant OLAP Viewer. When accessing OLAP cubes via ODBO, the SAS OLAP Server acts as an Open OLAP Provider, and SAS Enterprise Guide is an Open OLAP Consumer. SAS Enterprise Guide features drillable, interactive graphics and enables users to generate ad hoc queries for analytics based on cube data.

The OLAP Analyzer supports all of the functionality required to navigate through multidimensional data, add topic-specific business calculations, and extract

information from multidimensional sources for further analysis with advanced statistical procedures or data mining. With the OLAP Analyzer and SAS Enterprise Guide, you can do the following:

- access SAS OLAP cubes directly from the metadata definition. You can preview each cube's dimensions, measure its contents, and override the default view before accessing the cube.
- slice and dice data as needed to explore the information, as well as drill-through to underlying detailed data.

SAS Enterprise Guide provides the following features for OLAP cube access, visualization and manipulation:

- support for drilling, slicing and pivoting in a cube to explore the cube data. You can also drill-through to the underlying detailed data for a cube. With drill-through capability, you can navigate in your data from the most summarized levels to the most detailed levels. You can drill down on all members of a level or drill down on a specific member of the level. You can also drill in the graph view.
- calculation (calculated member or measure) support, including simple calculations, count analysis, relative contribution analysis and custom calculations (such as time series analysis).
- support for filtering that is based on a ranking function or on a range of values. You can now filter by member caption and member property. You can also create filters that can be used by multiple queries.
- support for totals, subtotals, and percent of totals calculations.
- multiple, independent views of a cube. Specific views on multidimensional information can be saved as bookmarks for easy reuse.
- surfacing of multidimensional information slices to other analytical SAS procedures for advanced analysis, including use in data mining procedures.
- surfacing of SAS OLAP data sources from the SAS OLAP Server or other third-party vendors supporting OLE DB for OLAP (for example, SAP BW and Microsoft Analysis Services).

For further information about SAS Enterprise Guide, see the Help for SAS Enterprise Guide.

SAS Information Map Studio

SAS Information Maps can translate business questions into the necessary MDX code to access SAS OLAP structures. SAS Information Map Studio is a Java application that enables data modelers and data architects to create and manage SAS Information Maps, which are business metadata about your physical data. Information maps enable you to surface your warehouse data in business terms that typical business users understand, while storing key information that is needed to build appropriate queries.

Information maps can be built from various data structures, including star schemas, snowflake schemas, normalized data structures, or OLAP cubes, providing business users with easy access to enterprise data. The SAS OLAP Server is required to create information maps on top of multidimensional data sources.

For more information about SAS Information Map Studio, see the Help for SAS Information Map Studio..

SAS Web Report Studio

SAS Web Report Studio is a Web-based application that enables you to create, view, and organize reports. You can use SAS Web Report Studio to perform these OLAP-specific tasks:

- drill and expand tables and graphs
- support ragged and unbalanced hierarchies
- pivot individual crosstabulation dimensions
- switch dimensions and measures with the Data Selection dialog box
- synchronize report components to display a common drill state or have them remain independent

In addition to OLAP-specific tasks, SAS Web Report Studio enables you to perform the following tasks:

Create reports
: Beginning with a simple and intuitive view of your data provided by SAS Information Maps (created in SAS Information Map Studio), you can create reports based on either relational or multidimensional data sources.

View and manipulate reports
: While viewing reports using a thin client (a Web browser), you can filter, sort, and rank the data that is shown in tables, crosstabulations, and graphs. With multidimensional data, you can drill down on data in crosstabulations and graphs and drill-through to the underlying data.

For more information about using SAS Web Report Studio, see the *SAS Web Report Studio User's Guide*,

SAS Add-In for Microsoft Office

The SAS Add-In for Microsoft Office is a Component Object Model (COM) add-in that extends Microsoft Office by enabling you to harness the power of SAS analytics and access data directly from Microsoft Excel, Microsoft Word, and Microsoft PowerPoint. Specifically, you can do the following:

- access and view SAS OLAP Cubes or any data source that is available from your SAS server. There is no size limit on the SAS data sources that you can open.
- filter your data using an intuitive user interface or using an advanced SQL editor.
- sort your data by an unlimited number of variables.
- refresh your data to incorporate any changes that were made to a data source that is saved on a server.

For more information, refer to the Help for the SAS Add-In for Microsoft Office..

Using a Cube with Third-Party Clients

Overview

The SAS OLAP Server exposes multidimensional data through OLE DB for OLAP interfaces. Supporting these industry-standard interfaces enables the SAS OLAP Server to integrate with third-party clients. The user interfaces for these clients vary widely.

Microsoft Excel Pivot Tables and Pivot Charts

Microsoft Excel provides the ability to view and manipulate SAS OLAP cubes in Excel pivot tables and Excel pivot charts. You can connect to a SAS Metadata Server and access a SAS OLAP Cube in Excel. Once a cube is accessed, you can select which measures and dimension to place on the pivot table or pivot chart. Specifically, you can perform the following OLAP cube-related tasks:

- assign measures and dimensions to the columns and rows of a pivot table or pivot chart
- drill-through to the cube's detail data and save that data into a separate sheet
- generate pivot charts based on the cube data already selected in the pivot table
- perform various data analysis tasks such as filtering, sorting and subtotaling

For further information about working with SAS OLAP Cubes in Microsoft Excel, see the cube building example “Implementing Drill-through to Detail Data in a SAS OLAP Cube” on page 272 “Implementing Drill-through to Detail Data in a SAS OLAP Cube” on page 272 .

11

Importing and Exporting SAS OLAP Cubes

Importing and Exporting SAS OLAP Cubes

Overview

Companies that create and use SAS OLAP cubes often have more than one environment that they store cubes on. A company might have an environment for developing and building cubes, another environment for testing cubes, and finally a production environment where cubes are accessed and queried by end users. If you have multiple environments for SAS OLAP cubes, you need the ability to copy cubes and their supporting files and objects from one environment to another.

Determining Which Tool to Use

When you choose to copy or move SAS OLAP cubes between SAS environments, choose the tool that is appropriate for your source SAS deployment.

- You can use the ExportCubes batch tool to export SAS 9.1.3 SP4 cubes.
- You can use the ImportCubes batch tool to import SAS 9.1.3 SP4 cubes to SAS 9.2 or a later version.
- Cubes that are SAS 9.2 or a later version can be moved to another version with the Export SAS Package and Import SAS Package functions that are found in SAS OLAP Cube Studio and SAS Management Console.

Note: You can use the ExportCubes batch tool only to move cubes out of SAS 9.1.3 SP4.

The following table shows the different export or import functions and in what SAS version they can be applied.

SAS Version	ExportCubes batch tools	ImportCubes batch tools	Export SAS Package	Import SAS Package
9.1.3 SP4	X			
9.2		X	X	X
9.3		X	X	X
9.4		X	X	X

In SAS 9.4, cubes that were built in SAS 9.2 or SAS 9.3 can be used without rebuilding them, as they maintain their physical status in metadata. However, some situations might require action on your part:

- If your SAS 9.2 or SAS 9.3 cubes were built using relative paths, you can move the physical files from one location to another and avoid rebuilding the cubes.
- If your SAS 9.2 or SAS 9.3 cubes were built in a common, network-accessible location, no action is required.
- If your SAS 9.2 or SAS 9.3 cubes are using new hardware in SAS 9.4, you must rebuild the cubes.

ExportCubes and ImportCubes Batch Tools

Overview

The ExportCubes and ImportCubes batch tools enable you to move your SAS 9.1.3 SP4 cubes to SAS 9.2 or later version. To use the batch tools, you must go to the DOS command line on your operating system.

ExportCubes Batch Tool

Overview

The ExportCubes batch tool is a command-line tool that enables you to export SAS 9.1.3 OLAP cube metadata. This tool exports the metadata for your cubes into individual XML files that are stored in a designated directory. The ExportCubes batch tool exports all cubes on the metadata server. It is available for use on Windows and is executed from a DOS command line.

When exporting cubes, you designate the output directory that you want to store the XML files to. A log file of the export process is automatically generated in the user's SAS application data logs folder and is named `ExportCubes.log` by default. Additional log files are assigned the default name and are appended with a number. The SAS application data logs folder is located in one of the following locations:

- For pre-Vista Windows: `C:\Documents and Settings\user-id\Application Data\SAS\Logs`
- For Windows 7 and Windows Vista: `C:\Users\user-id\AppData\Roaming\SAS\Logs`

- For UNIX and z/OS: `~/.SASAppData/SAS/Logs`

You can also designate whether security should be exported with the cube metadata or not. When you export cube metadata, the name of the XML file is generated from a combination of the name of the cube's OLAP schema (if there is one) and the name of the cube. If the OLAP schema or cube name contains characters that are not allowed for a filename on that particular operating system, then those characters are changed according to the native Java interface for that operating system.

When you export your cube metadata, the output directory is populated with an XML file for each extracted cube and a log file. All cubes in the Foundation and Custom repositories are exported. However, cubes in Project repositories are not exported and are ignored.

Note: The ExportCubes batch tool is supported in a hot fix that is applied to SAS 9.1.3 SP4. Cube metadata that is exported with the ExportCubes tool can then be imported in SAS 9.2 or later version with the ImportCubes batch tool. In addition, cube metadata that is created with a version of SAS that is later than SAS 9.1.3 SP4 cannot be exported with the ExportCubes batch tool. If attempted, an error message is printed to the log.

Syntax

To use the ExportCubes batch tool, go to the DOS command line on your operating system. The batch tool can be found in the following directory path:

`SAS-Installation-directory\sasmanagementconsole\9.1.3\commands`

You can enter the ExportCubes command with the needed options.

ExportCubes - <option(s)>

The following example shows the ExportCubes command:

```
ExportCubes -user "x2345\sasxxx" -password "Passwordxxx" -host
"X2345.xy.abc.com"
-port 8561 -extractSecurity -outputDirectory c:\myextractedcubes
```

Table 11.1 *ExportCubes Options*

Option Name	Description
-extractSecurity	Specifies to extract security metadata for the cube. If not specified, security metadata is not extracted.
–host	Specifies the server to connect to. The metadata connection can be specified by a profile or the server, port, user, and password options.
-outputDirectory	Specifies the output directory. The cube metadata and log is written to this directory. If the directory exists, then a number (starting with 1) is appended to the end of the directory name until the name is

Option Name	Description
	unique. By default, the log file is named ExportCube.log.
-password “pw”	Specifies the password for the user. The metadata connection can be specified by a profile or the server, port, user, and password options.
-port number	Specifies the port for the server. The metadata connection can be specified by a profile or the server, port, user, and password options.
-profile filename	Specifies the workspace file to use for metadata connection information. The metadata connection can be defined by specifying the server, port, user, and password options or by specifying a metadata connection profile. This option can be used in place of the server, port, user, and password options.
-user “username”	Specifies the user to connect to the server with. The metadata connection can be specified by a profile or the server, port, user, and password options.

ImportCubes Batch Tool

Overview

The ImportCubes batch tool is a command-line tool that enables you to import SAS 9.1.3 SP4 OLAP cube metadata files that have been exported using the ExportCubes batch tool on your SAS 9.1.3 SP4 environment. The ImportCubes batch tool is supported in SAS 9.2 and later versions. Cubes that are exported with the ExportCubes batch tool must also be imported with the ImportCubes batch tool. The ImportCubes batch tool is compatible only with cubes that were exported with the corresponding ExportCubes batch tool on your SAS 9.1.3 SP4 environment. The ImportCubes batch tool is not compatible with cubes that were exported from SAS 9.1.3 SP4 using the SAS OLAP Cube Studio Export Cube function.

The metadata for a cube is exported as an XML file. With the ImportCubes batch tool, you can specify the location of multiple exported SAS 9.1.3 XML files and import them to your SAS OLAP environment. The SAS 9.1.3 SP4 cube metadata is imported and added to the SAS Metadata Server . After the metadata files are imported, you can then rebuild your SAS 9.1.3 SP4 cubes. You can use SAS OLAP Cube Studio to build the physical data for the cubes.

A log file of the import process is automatically generated in the user's SAS application data logs folder. This log file is named `ImportCubes.log` by default.

Additional log files are assigned the default name and are appended with a number. You can also specify a directory for the ImportCubes.log file.

The ImportCubes batch tool is included with the standard SAS OLAP installation and is available for use on Windows. It is executed from a DOS command line. When you execute the ImportCubes command, all the cube XML extract files in a directory are imported. The cubes are imported into the repository from which they were exported. If that repository does not exist on the target metadata server, then the cubes are imported into the Foundation repository and this is noted in the ImportCubes log file.

If needed, you can substitute file paths during the import of your cube metadata. To substitute file paths during the import, each XML file where you want to substitute file paths must be edited. To change file paths for a cube, you must edit the Substitutions section of the XML file. Specifically, you must change the string following the tag “DefaultValue=” to the directory path that you want. Do this for each XML file that you want to change, and then run the ImportCubes batch tool. The following example shows the Substitutions section of an XML file.

```
<Substitutions>
<Property Id="A5G2Y32Q.AI000EOC" TLObjN="1" Name="INDEXPATH"
   DefaultValue="c:\indexfilesAggrA"/>
<Property Id="A5G2Y32Q.AI000EOD" TLObjN="1" Name="DATAPATH"
   DefaultValue="c:\partitionsAggrA"/>
<Directory Id="A5G2Y32Q.B00000RU" TLObjN="1" Name="c:\v9cubes"
   DirectoryName="c:\v9cubes"/>
<Property Id="A5G2Y32Q.AI000EO9" TLObjN="1" Name="WORKPATH"
   DefaultValue="c:\v9cubes"/>
<Property Id="A5G2Y32Q.AI000EOA" TLObjN="1" Name="INDEXPATH"
   DefaultValue="c:\globalindex1"/>
<Property Id="A5G2Y32Q.AI000EOB" TLObjN="1" Name="DATAPATH"
   DefaultValue="c:\globalpartitions1"/>
</Substitutions>
```

Cube Connection Points

For each cube, the connection points that the ImportCubes batch tool expects to find must exist in the target repository. If a table, column, UniqueKey, or OLAP schema connection cannot be found in the target metadata server, the import of the cube fails and an error message is written to the ImportCubes log file. For the OLAP schema connection of a cube, if the OLAP schema does not exist on the target metadata server, then the cube is imported and not associated with any OLAP schema.

Syntax

To use the ImportCubes batch tool, go to the DOS command line on your operating system. The batch tool can be found in the following directory path:

SAS-Installation-directory\SASPlatformObjectFramework\9.2

You can enter the ImportCubes command with the needed options.

ImportCubes - <option(s)>

The following example shows the ImportCubes command:

```
ImportCubes  -user "sasadm" -password "Password01" -host
"X7891.xy.abc.com"
-port 8561  -inputDirectory  c:\myextractedcubes
-log c:\mylogfiles
```

Table 11.2 *ImportCubes Options*

Option Name	Description
-host	Specifies the server to connect to. The metadata connection can be specified by a profile or the server, port, user, and password options.
-inputDirectory	Specifies the input directory that contains the XML cube extract files from the 9.1.3 ExportCubes batch tool. This option is required.
-log directory	Specifies the directory where the log file named ImportCubes.log is created. If this option is not specified, the log file is automatically created in the user's SAS application data logs folder. This can be in one of the following locations: ■ For pre-Vista Windows: `C:\Documents and Settings\user-id\Application Data\SAS\Logs` ■ For Windows 7 and Windows Vista: `C:\Users\user-id\AppData\Roaming\SAS\Logs` ■ For UNIX and z/OS: `~/.SASAppData/SAS/Logs`
-password “pw”	Specifies the password for the user. The metadata connection can be specified by a profile or the server, port, user, and password options.
-port number	Specifies the port for the server The metadata connection can be specified by a profile or the server, port, user, and password options.
-profile filename	Specifies the workspace file to use for metadata connection information. The metadata connection can be defined by specifying the server, port, user, and password options or by specifying a metadata connection profile. This option can be used in place of the server, port, user, and password options.
-user “username”	Specifies the user to connect to the server with. The metadata connection can be specified by a profile or the server, port, user, and password options.

Export SAS Package and Import SAS Package

Overview

A SAS OLAP cube contains a physical files component and a metadata component. Both of these components must be addressed and kept in-sync when copying or moving a cube. When copying or moving SAS cubes between SAS systems, the physical files that compose a cube are copied with standard operating system functions and line commands. The cube metadata is then copied with Export SAS Package and Import SAS Package functions that are found in SAS OLAP Cube Studio and other SAS Intelligence Platform products.

SAS Packages - Copying the Cube Metadata Registration for a SAS OLAP Cube

Part of copying a cube includes copying the metadata registration for the cube. Begining with SAS 9.2, SAS OLAP cubes and their supporting objects can be exported and imported as a group in a SAS package (SPK) file. You can use the Export SAS Package function to create an (SPK) file. This package file can then be imported into another system with the Import SAS Package function. The Export SAS Package and Import SAS Package functions are part of the SAS Intelligence Platform promotion tools. These tools enable you to promote individual metadata objects or groups of objects from one metadata server to another, or from one location to another on the same metadata server.

Export SAS Package

In SAS OLAP Cube Studio, you can create a SAS package to export to another system. In the tree view, determine the object or objects that you want to export. Use the CTRL key to select multiple objects. These objects can include cubes, jobs, libraries, tables, folders, or OLAP schemas. You can also select multiples of the same object. For example, you can export two cubes in the same package.

Note: A cube must be exported with its corresponding cube job.

After you select the needed objects, select **Export SAS Package** from the **File** menu or from the cube's context menu. This opens the Export SAS Package wizard. To export a SAS package, perform the following tasks:

1. On the Welcome page of the wizard, specify the name of the SAS package that you are exporting cube objects to. You can also select the option **Include dependent objects when retrieving initial collection of objects**. This signals the wizard to automatically include any dependent objects for the item(s) you initially selected in the tree view.
2. On the Select Objects to Export page of the wizard, select the objects to include in the exported SAS package.
3. On the Summary page of the wizard, confirm the objects to include in the SAS package. Select **Next**. The SAS package is created.
4. On the Export Complete page, you can view the export log and verify that the package was created.

Be aware that SAS OLAP cubes are dependent on a number of other objects. These include, in particular, cube jobs, tables, libraries, and OLAP schemas. For a successful import, all these objects need to be available on the target system. You must know whether these objects are already available in the target system. However, you can export the objects now and verify their availability later, when you import the package. If you are selecting a cube to export, always select the cube together with its job. The export does not work if the job is not present.

Import SAS Package

After you have exported the SAS package, you can import the package on the target system. Switch to your target system and make sure that the exported SAS package file and the copied cube files are available there.

At this point, you can open SAS OLAP Cube Studio and select a folder in the tree view. This selected folder is where you are importing the package to. If the OLAP schema that you need is on the target system, you can import your cube and related objects into the folder that contains your OLAP schema.

Note: The Import SAS Package function is available only when you select a folder in the tree view.

Select **Import SAS Package** from the **File** menu or from the cube's context menu. The Import SAS Package wizard opens. To import a SAS package, perform the following tasks:

1. On the Welcome page of the wizard, specify the name of the SAS package that you are importing. On this page, you can select to include the objects' access control templates. In addition, you can select to import all the objects in the package or only those that are new and do not exist in the destination folder.
2. On the Select Objects to Import page, you can select the objects from the package that you want to import. When you click an object in the list of objects to import, any available options for that object are available on the **Options** tab.

An important option to verify when importing a SAS OLAP cube is the cube option **The physical cube will need to be built after the import process**. This indicates that the physical cube files do exist on the target system. By default, this option is automatically selected. If you select this option, deselect any objects that are already available on the target system. Also, deselect any objects that you don't want to create in the target folder. The physical cube will be built after the cube is imported into the target system.

Note: A cube must be imported with its corresponding cube job.

3 After you have selected the objects to import and build, the wizard identifies those objects that you must establish metadata definitions or connection points for. The About Metadata Connections page lists the metadata objects that you must define. This list is determined by the objects that you selected on the previous page. The Import SAS Package wizard enables you to specify values for the target location that correspond to values from the source location. Depending on what you import with the cube, this can include the following objects:

- OLAP schema (This can have the same name as the original OLAP schema, but reside in a different folder.)
- tables (These include fact tables, dimension tables, drill-through tables, or aggregation tables.)
- libraries
- SAS Application Server
- directory paths (These include the cube path and optionally, data paths, index paths, and library paths.)

4 Next, complete the wizard page for each metadata object identified on the About Metadata Connections page. When you have finished entering the connection point information, you can view the Summary page of the wizard. This page lists the objects that are being created and the import objects that are being mapped on the target system.

User Privilege and Permission Considerations

It is recommended that the person performing the cube export or import have SAS Administrator priviledges and preferably be an unrestricted user. This is because other users might have restrictions on parts of the cube that could result in a partial cube being exported and imported. It is also important that the identities, groups, and permissions for the imported data be set to correspond to the target environment security settings. Security settings can be created using the **Authorization** tab on the cube Properties dialog box.

Creating Connection Points

Certain metadata must exist on the target system prior to importing a cube. If a metadata connection object is not found, then the import fails. The following metadata connection points must exist:

Table 11.3 *Connection Points*

Object	Connection Points	Details
Cube	Job(cube), OLAP schema, tables, directory path object	A cube must be imported with its corresponding job(cube).
Job(cube)	Cube	A cube must be imported with its corresponding job(cube).
OLAP Schema	SAS Application Server and its logical OLAP server	Only one OLAP schema can be associated with a SAS Application Server.
Library	SAS Application Server and directory path object	The target application server that you select determines the SAS libraries that are available.
Tables	Library	If there is a matching SAS library on the target system, then that SAS library is selected by default. This occurs if the target system library has the same library name and SAS Application Server name as the original SAS library.

Multi-Language Cubes

You can export and import multi-language cubes. However, only the dimension tables for the server language (the first language in the UDT statement) are verified for registration. You must ensure that all the tables are present. If one of the tables is missing when the cube is imported, the import is still successful, but the cube might not rebuild correctly.

Manually Copying Physical Files for a SAS OLAP Cube

SAS OLAP Cube Files

When you copy a SAS OLAP cube between environments, you can choose to copy the physical files that are part of the cube (in order to not rebuild these files on the target system). To do this, you copy or move all of the files from the source cube directory to the target cube directory. You can use standard operating system functions to copy or move the files. It is important to check and verify that the target files have the same operating system permissions as your original files. In addition, when you are copying files between environments, note that you can copy SAS OLAP cube files only among systems that have the same data representation. You cannot move between 32–bit and 64–bit systems, and you cannot move between Windows and UNIX. Manually copying SAS OLAP cube files involves the following components:

- cube directories
- cube header files
- MOLAP aggregation tables
- ROLAP data tables

Note: Data tables that are used in ROLAP cubes can be included in the exported SAS package.

Creating Cube Directories

Before you copy the files for a cube, you must create a directory for the cube on the target system. The path that you create the directory in is known as the root path for the cube and should have the same name as the cube. The cube's path is specified with either of the following methods:

In SAS OLAP Cube Studio
: When creating or editing a cube in the Cube Designer wizard, enter a file path in the **Path** field on the Cube Designer – General page.

Using PROC OLAP
: Include the PATH= option in the PROC OLAP statement.

For example, if you create the cube OrionStar in the cube path `\\myserver\testolap\testcube`, then the cube root path is `\\myserver\testolap\testcubes\OrionStar`.

When you copy the OrionStar cube to your target cube path, create a directory with the cube's name. For example, if you want to copy or move the cube to

otherserver\prodolap\prodcubes, you would create the directory **otherserver \prodolap\prodcubes\OrionStar**.

If the cube has aggregations or indexes stored in other directories, then these files must also be copied or moved to a directory with the same name. You can use the DATAPATH= and INDEXPATH= options to specify what directories the aggregations are stored in.

Manually Copying the Header Files for Cube Generations

The header files for each generation of the cube are located in subdirectories in the cube root path. They are named gen0000, gen0001, and so on. For example, if the cube OrionStar is created in the cube path **\\myserver\testolap\testcubes**, then the cube header files are stored in the path **\\myserver\testolap\testcubes \OrionStar\gen0000**. If you have made incremental updates to the cube, you might have more than one genNNNN directory. Or your genNNNN directory might have a numeric suffix other than 0000. Copy or move all the genNNNN subdirectories with their contents to the target cube root directory.

Manually Copying the MOLAP Aggregation Tables

SAS OLAP cubes store aggregated data values in tables that have the same file structure as SAS SPD Engine tables. These tables consist of a number of different files and can be broken into partitions. By default, all the component files and partitions are stored in the cube root directory. However, you might have chosen to distribute the MOLAP aggregation storage over multiple locations. You can specify distributed file storage locations with either of the following methods:

In SAS OLAP Cube Studio
: When creating or editing a cube in the Cube Designer wizard, you can define the storage location for one or more aggregations. On the Aggregations page of the wizard, select the **Advanced** button to open the Performance Options dialog box. From here you can define the storage location for all aggregations or a single aggregation. On either the **Default** tab or the **Aggregations** tab, enter the file path in either of the following fields:

  - **Physical path to indexes**
  - **Physical path to aggregation tables**

Using PROC OLAP
: Use the DATAPATH= and INDEXPATH= options in the PROC OLAP statement or in the individual AGGREGATION statements.

Copy your MOLAP aggregation tables by using operating system utilities into the target directories. Be sure to copy the files from all locations, if you have distributed your cube aggregation storage over multiple directories.

Manually Copying Rolap Files

ROLAP tables are used as data sources for SAS OLAP cubes and can be stored in many formats. These formats include SAS tables, SPD Engine tables, SPD Server tables, and tables in external RDBM systems. When you are building SAS OLAP cubes, ROLAP tables can be used as the following data sources:

- drill-through tables
- aggregation tables
- input data (if you used the NO_NWAY option)
- format catalogs (used by any of the above)

You can use PROC COPY to copy or move SAS tables, SPD Engine tables, and SPD Server tables. To copy or move tables in external RDBM systems, use standard operating system functions. When you copy or move any of these files, you should verify that your target SAS OLAP Server has access to those files. In addition, you need to change the LIBNAME specifications to point to the new file locations. Librefs for an OLAP server are allocated by using an AUTOEXEC file during the SAS OLAP Server invocation or by using pre-assigned libraries.

Validating Data After It Is Moved

After migration or promotion of your data is complete, the cubes must be rebuilt in SAS 9.4. In addition, other tasks must be completed for SAS OLAP Server and SAS OLAP Cube Studio. The following tasks must be performed:

- In SAS Management Console, ensure that all SAS OLAP Servers and OLAP schemas are available.
- In the Server Manager, validate the connections to the SAS OLAP Servers.
- Use SAS OLAP Cube Studio to build a test cube.
- Use the Cube Viewer in SAS OLAP Cube Studio to validate the structure of the test cube.
- Build cubes and validate their structure on all other SAS OLAP servers.

Cube Promotion and Migration Resources

In addition to the export and import cube functions discussed in this chapter, see the *SAS Intelligence Platform: Migration Guide* for information about promoting and migrating your SAS data:

Chapter 12

OLAP Procedure

Overview: OLAP Procedure

What Does the OLAP Procedure Do?

The OLAP procedure is one of the SAS tools that you can use to create, update, and delete cubes and shared dimensions.

You can also use the Cube Designer wizard to generate and execute OLAP procedure code. The Cube Designer wizard can be launched from SAS Data Integration Studio and SAS OLAP Cube Studio. Help on using the wizard to build cubes is available from within both applications.

The OLAP procedure enables you to perform the following tasks:

- build cubes with ragged hierarchies
- create multilingual cubes
- define dimensions that are shared between two or more cubes
- rename cubes
- change nonstructural cube elements
- specify options that optimize cube creation
- enhance query performance by generating aggregations
- specify data set options on detail, fact, dimension, and drill-through tables
- add geographical connections based on Esri mapping data
- design dimensions that have more than one hierarchy
- define global calculated members and named sets

Syntax: OLAP Procedure

PROC OLAP <*options*>;

METASVR OLAP_SCHEMA='*schema-name*' < *options*>;

DIMENSION *dim-name* HIERARCHIES=(*hier–nam ... hier-nameN*) <*options*>;

USE_DIMENSION [*dim-name* | *"/folder/dim-name"*] [OLAP_SCHEMA=*schema-name*]
FACTKEY=*fact-key column*;

HIERARCHY *hier-name* LEVELS=(*level–name1* <*level–name2 ...level-nameN*>) <*options*>;

LEVEL *level-name* <*options*>;

PROPERTY *prop-name* LEVEL=*level-name*< *options*>;

MEASURE *measure-name* STAT=*statname* <*options*>;

AGGREGATION *level-name* <*level-name2 level-name3 ...level-nameN*> / <*options*>;

DROP_AGGREGATION *level-name1* < *level-name2 ...level-nameN*> / NAME=*aggregation-name*;

DEFINE MEMBER | SET *'member-or-set-name'* AS *'mdx-expression'* ;

UNDEFINE MEMBER | SET *'member-or-set-name'* ;

USER_DEFINED_TRANSLATIONS *locale* <*locale2 ...localeN*> ;

REORGANIZE_LEVEL | REORG_LEVEL *level-name*

PROC OLAP Statement

Specifies the input data source, cube name, and path.

Syntax

PROC OLAP <*options*>;

Details

Overview

The PROC OLAP statement can be used to do the following:

- create cubes and shared dimensions
- specify options that might improve query performance
- delete cubes and shared dimensions
- specify global settings for handling missing hierarchy members in ragged and unbalanced hierarchies
- perform actions on existing cubes
- change nonstructural cube elements like captions and descriptions
- update cube data
- coalesce updated aggregations

Options

ADD_DATA
specifies an incremental update to a cube. The ADD_DATA option enables you to add new members and data to a cube. You can do a managed or an in-place update of the cube. The new data that the cube is updated with is specified with the DATA= option. Here is an example of the ADD_DATA option used with PROC OLAP:

```
proc olap
    data=mylib.newdata cube=cubeA add_data
        outcube=cubeB outschema=testSchema;
    metasvr host="myhost" port=8561 repository=myrepository
        olap_schema=prodSchema;
run;
```

The ADD_DATA option requires that either UPDATE_IN_PLACE, OUTCUBE=, or OUTSCHEMA= also be specified. The DATA= or FACT= option is required to specify a table with the new input records.

Note: You cannot use ADD_DATA with cubes that have been created with the NONUPDATEABLE option.

ASYNCINDEXLIMIT= *n*
specifies a limit on the number of indices that will be created in parallel during the cube build process. By default, all indices for an aggregation in the cube are created asynchronously. The number of indices built is based on the number of hierarchies present in the aggregation. Therefore, if your cube contains a large number of hierarchies and your machine has limited resources, you might find that limiting the indices created asynchronously can result in better cube build performance. Use a value greater than zero to set this limit. The default value is 0. There is no limit, as all indices for an aggregation are created asynchronously.

Note: The NWAY is the largest aggregation in the cube and contains data for ALL hierarchies.

COALESCE_AGGREGATIONS
specifies that all aggregations in a cube will be coalesced. When a cube is updated, multiple aggregation partitions (separate SPD Engine tables) are produced by one or more cube update (ADD_DATA) operations. The more partitions an aggregation has, the slower the access time will be. To prevent reduced query performance on a cube, these aggregation partitions can be coalesced (combined) back into a single aggregation table. Coalescing updated aggregations periodically keeps the number of aggregation partitions at a minimum. When an aggregation is coalesced, all existing partitions are grouped together to create a single SPD Engine table containing data from all partitions. This becomes the new aggregation table and the old partitions are deleted. Only aggregations with more than one partition are coalesced. All others are ignored (and a warning message is printed to the log).

Only MOLAP aggregations (those aggregations created by PROC OLAP) can be coalesced. No ROLAP aggregations are coalesced. If a ROLAP aggregation is specified in a COALESCE_AGGREGATION statement, that statement is ignored and a warning message is printed to the log.

COMPACT_NWAY
specifies that the cube build will include an additional summarization step that is designed to decrease the size of the NWAY aggregation and improve viewing performance. The amount of improvement depends on the nature of the data. The cubes that improve the most are those that have the largest number of rows that can be included in the additional summarization step.

Candidates for compaction are cubes that are built from star schema, where the cube does not define levels for all columns in all dimension tables. This can result in a fact table that contains many rows that belong to the same leaf member of a given hierarchy. These are the rows that are summarized to decrease the size of the NWAY aggregation.

For example, assume that a cube is built from a star schema that contains a Time dimension table. The Time table contains columns for year, quarter, month, and day, along with a primary key column. If the cube is defined so that the day column is not specified as a level of a time hierarchy, then there are up to 31 key values that refer to each unique combination of year, quarter, and month. Together, these key values define a unique leaf member of that hierarchy. These are the values that are summarized at build time.

The amount of compaction in the NWAY aggregation is determined by the number of source rows that can be summarized. The number of summarized rows depends on the number of unique key values in the fact table that refer to the same leaf member of a hierarchy. Another compaction factor is the number of rows in the fact table that contain unique combinations of keys; these rows are not compacted.

COMPRESS | NOCOMPRESS
specifies whether to store the aggregation tables in a compressed format on disk. NOCOMPRESS is the default setting. This option applies to the automatically created NWAY and all aggregations that do not explicitly specify a COMPRESS option in the AGGREGATION statement.

CONCURRENT= *n*
specifies the maximum number of aggregations to create in parallel. This option does not apply to the NWAY aggregation, which is always built first (unless the NO_NWAY option is set).

The default value is 2, which is based on the results of a special algorithm that takes into consideration the number of aggregations that are being created and the number of processors that are available. The algorithm assumes that CPU resources should be reserved for creating aggregation indexes.

So that each built index has a fair share of the assigned INDEXSORTSIZE memory, INDEXSORTSIZE is divided by the CONCURRENT value. The value of INDEXSORTSIZE should give each concurrent index build enough memory to at least hold a table PARTSIZE. For best performance, INDEXSORTSIZE divided by CONCURRENT should be greater than PARTSIZE.

CUBE=*cube-name* | *folder-path/cube-name* <(*cube*>
specifies a valid SAS name for the cube to be created, renamed, or updated. The folder path that is specified is a metadata folder path. It exists only in the metadata. In SAS 9.4, cubes are identified both by their location in an OLAP schema and by their association with a metadata folder. By default, if no *folder-path* is given, the cube will be associated with the folder that contains the OLAP schema. When you specify a metadata folder, you can add the metadata type (CUBE) after the cube name. For naming guidelines, see “Naming Guidelines and Rules for the SAS OLAP Server ” on page 51.

Note: The cube name must be unique within the OLAP schema and the metadata folder. The specified metadata folder path must exist.

Here are some examples of the CUBE= option.

- Example 1

```
proc olap cube='MYCUBE' /* other options */;
metasvr ... olap_schema='SASApp - OLAP Schema'; /*
further statements */ run;
```

This example creates a cube named MYCUBE in the OLAP schema SASApp - OLAP Schema and also surfaces the cube in that schema's metadata folder. It is located by default in **`/Shared Data/SASApp - OLAP Schema`**.

- Example 2

```
proc olap cube='/Shared Data/Cubes/MYCUBE(cube)'
/* other options */; metasvr ... olap_schema='SASApp - OLAP Schema';
/* further statements */ run;
```

This example creates a cube named MYCUBE in the OLAP schema SASApp - OLAP Schema and also surfaces the cube in the metadata folder `/Shared Data/Cubes`.

CUBETABLELIBREF=*lib*
specifies the libref containing the translated caption tables. This has the same functionality for the cube as the DIMTABLELIBREF has to the dimension.

CUBETABLECAPPREF=*caption-table-prefix*
specifies the member prefix for the translated cube CAPTION tables. The member prefix is the prefix of the data set name. The suffix of the name is provided by the USER_DEFINED_TRANSLATIONS statement. For example, if the caption member prefix is `carscubecap` and the suffix is `da_DK`, then PROC OLAP looks for a data set named `carscubecapda_DK.sas7bdat` in the library that is specified by the CUBETABLELIBREF= option. It is used in conjunction with the USER_DEFINED_TRANSLATIONS statement. This specification is optional. If caption data sets are specified but no caption is found, the default behavior is to use the captions from the default language.

DATA | FACT=*dsname*
specifies the data source for the cube. The unsummarized data source can be any SAS data file, including files that are supported by SAS/ACCESS software engines. If you load the cube from a star schema, then the *dsname* is the name of the fact table that contains the analysis variables from which to derive the measures for the cube. The fact table must also contain fact keys that correspond to dimension tables in the star schema.

You can also provide data set options along with DATA | FACT=. Options are stored within the cube and reapplied when the data is accessed at run time. For more information, see *SAS Data Set Options: Reference*.

If you load the cube from a star schema, then you must use the DIMENSION statement to do the following:

- specify the dimension table name (the DIMTBL= option)
- specify the dimension (primary) key column (the DIMKEY= option)
- specify the column (foreign key) in the fact table that corresponds to the dimension key column (the FACTKEY= option)

DATAPATH=(*'pathname' ...'pathnameN'*)
specifies the location of one or more partitions in which to place aggregation table data. The data is distributed by cycling through each partition location according to the partition size (set using the PARTSIZE= option). For example, suppose that you specify the following:

```
DATAPATH=('C:\data1' 'D:\data2')
```

Then PROC OLAP places the first partition of each aggregation table into directory `C:\data1`, the second partition of each table into directory `D:\data2`, the third partition of each table into `C:\data1`, and so on. It is also possible to have aggregation tables that use fewer than the specified number of partitions. For example, your cube might contain an aggregation table that fits entirely into `C:\data1`.

Note: This option applies to the automatically created NWAY and all aggregations that do not explicitly specify a DATAPATH= option in the AGGREGATION statement.

The default location is the cube subdirectory of the location that is specified by the PATH= option in the PROC OLAP statement.

DELETE
deletes the physical cube that is specified with the CUBE= option. It also deletes the cube's registration, which is stored in the metadata repository. If either the physical cube, its registration, or both are not present, then the DELETE option behaves as explained in the following table:

Note: The use of the DELETE option will remove all information about a cube, including security information and information maps.

Table 12.1 *How the DELETE Option Behaves If the Physical Cube or Its Registration Is Not Present*

Physical Cube Exists	Registration Exists	Behavior of DELETE Option
No	Yes	The physical cube is not deleted. The registration is deleted. If there is a registration, and you use the DELETE option, the registration is always deleted and you cannot recreate the cube from the registration. You can recreate the cube only from the registration when you use the DELETE_PHYSICAL option.
No	No	Fails because there is nothing to delete.
Yes	No	Fails because the cube cannot be located without its registration information. You must manually delete the cube files.

DELETE_PHYSICAL
deletes the physical cube that is specified with the CUBE= option but leaves the cube registration intact. This enables you to build a new cube based on the saved cube registration. With the DELETE_PHYSICAL option, the cube registration is maintained in the metadata. As a result, you can change the registration of the cube without changing the cube name or file path. The physical cube must be removed before that cube can be rebuilt.

Note: The use of the DELETE_PHYSICAL option is preferable when rebuilding a cube, because it preserves information about a cube, including security information and information maps.

If either the physical cube, its registration, or both are not present, then the DELETE_PHYSICAL option behaves as explained in the following table:

Table 12.2 How the DELETE_PHYSICAL Option Behaves If the Physical Cube or Its Registration Is Not Present

Physical Cube Exists	Registration Exists	Behavior of DELETE_PHYSICAL Option
No	Yes	A warning message is given that there is no physical cube to delete.
No	No	Fails because there is nothing to delete.
Yes	No	Fails because the cube cannot be located without its registration information. You must manually delete the physical cube files.

DESC | DESCRIPTION=*'cube-description'*
specifies the characters to be stored as descriptive text. The number of characters that can be used for the *cube-description* is unlimited.

DRILLTHROUGH_TABLE | DT_TABLE | DT_TBL=*table-name*
specifies an optional drill-through table. Drill-through tables can be used by client applications to provide a view from processed data into the underlying data source. You can specify the DATA | FACT= table or a different table that includes the columns used for the levels and measures in the cube. A minimum subset of the columns used for the levels is required. You can also specify data set options with this option. Options are stored within the cube and reapplied when the data is accessed at run time. For more information, see *SAS Data Set Options: Reference*.

DTLIBREF=*libref*
specifies the SAS library that contains the drill-through tables that you are defining for your cube. The DTLIBREF= option is needed for locale-specific multilingual cubes.

DTMEMPREF=*member-prefix*
specifies the member prefix for your translated drill-through tables.

DTMEMPREFOPTS=*table options*
specifies the table (or data set) options that you can use when opening drill-through tables. The following example opens only the first 400 records in any drill-through table:

```
PROC OLAP CUBE=MLSCARS DTLIBREF=OLAPLIB DTMEMPREF=CARS DTMEMPREFOPTS="obs=400";
```

EMPTY_CHAR='*string*'
specifies the quoted text string that identifies members of character levels that are to be skipped or disregarded. Members are skipped in order to create ragged or unbalanced hierarchies, as described in "Defining Ragged and Unbalanced Hierarchies for a Dimension" on page 71. The maximum length of the quoted string is 256 characters.

To be skipped, a member in a character level must have a caption whose value matches the value of the EMPTY_CHAR= option. For example, if a member in a character level is skipped, and if the caption of that member is **`Empty`**, then the EMPTY_CHAR= option is specified as follows:

```
empty_char='Empty'
```

When specified in the PROC OLAP statement, the EMPTY_CHAR= option can be overridden by the EMPTY_CHAR= or IGNORE_EMPTY options in a HIERARCHY statement or by the EMPTY= or IGNORE_EMPTY options in a LEVEL statement. To skip members in numeric levels, use the EMPTY_NUM= option.

EMPTY_NUM='*string*'
specifies the quoted text string that identifies members of numeric levels that are to be skipped or disregarded. Members are skipped in order to create ragged or unbalanced hierarchies, as described in "Defining Ragged and Unbalanced Hierarchies for a Dimension" on page 71. The maximum length of the quoted string is 256 characters.

To be skipped, a member in a numeric level must have a caption whose value matches the value of the EMPTY_NUM= option. For example, if a member in a numeric level is skipped, and if the caption of that member is `Empty`, then the EMPTY_NUM= option is specified as follows:

```
empty_num='.'
```

When specified in the PROC OLAP statement, the EMPTY_NUM= option can be overridden by the EMPTY_NUM= or IGNORE_EMPTY options in a HIERARCHY statement or by the EMPTY= or IGNORE_EMPTY options in a LEVEL statement. To skip members in character levels, use the EMPTY_CHAR= option.

Note: If there is no format that is associated with the member value, then BEST12 is used as the format.

ESRI_MAP_SERVER=*MapServerName*
specifies the Esri map server to which this cube should be linked. The map server must already be defined on the metadata server. The Esri map server must be an ArcGIS server defined on the metadata server. The name will be unique across all ArcGIS servers defined on the metadata server. There can be only one map server per cube.

ESRI_REPLACE
indicates that any existing associations to Esri metadata should be replaced with those specified in this PROC OLAP session. If no Esri options are provided, all existing linkages are removed. If ESRI_REPLACE is not specified and there is a conflict between existing associations and those specified in this session, an error results and no associations are added or changed.

FORCE
when specified, enables you to delete an existing cube and then immediately rebuild the cube. This includes building a cube with the short form of PROC OLAP. If the cube already exists in the specified schema, the cube is deleted prior to building the new cube. In addition, if a metadata folder path is specified, the cube must exist in the specified folder or the delete fails.

Note: If the cube build fails, the deleted cube is not restored. This is because the DELETE or DELETE_PHYSICAL option is committed prior to the start of the build. There is no rollback of the original cube.

IGNORE_MISSING_DIMKEYS= |
when specified while building a cube from a star schema, causes SAS to ignore an error condition, log the error, and continue building the cube. The error condition is detected when the fact table contains foreign key values that are not present in one of the contributing dimension tables. By default, and when this option is not specified, any missing dimension keys stop the build of the cube. When IGNORE_MISSING_DIMKEYS=TERSE is specified, the cube build continues and the fact table row with the missing key is ignored (it is not built into the cube). The SAS log receives an entry that lists the total number of key values that are missing from each dimension table. Specifying a value of VERBOSE produces the same behavior, except that the log receives additional details; the missing keys are listed for each dimension table.

INDEX | NOINDEX
specifies whether to create the aggregations with indexes. For faster cube creation and adding and deleting aggregations, you can set this option to NOINDEX. However, the lack of indexes will adversely affect query performance. The default value is INDEX.

Note: This option applies to the automatically created NWAY and all aggregations that do not explicitly specify an INDEX option in the AGGREGATION statement.

INDEXPATH=(*'pathname' ...'pathnameN'*)
specifies the locations of the index component files that correspond to each aggregation table partition as specified by the DATAPATH= option. The default value is the cube subdirectory of the location that is specified by the PATH= option in the PROC OLAP statement. This option applies to the automatically created NWAY and all aggregations that do not explicitly specify an INDEXPATH= option in the AGGREGATION statement.

Note: Indexes are not created for aggregations that have fewer than 1,024 records.

INDEXSORTSIZE=*n*
specifies the amount of memory in megabytes that is available when aggregations are being created. The default value is the system's available memory. So that each built index has a fair share of the assigned INDEXSORTSIZE memory, INDEXSORTSIZE is divided by the CONCURRENT value. The value of INDEXSORTSIZE should give each concurrent index build enough memory to at least hold a table PARTSIZE. For best performance, INDEXSORTSIZE divided by CONCURRENT should be greater than PARTSIZE.

MAX_RETRIES =
specifies the number of times PROC OLAP will attempt to reconnect to the metadata server after a connection is lost. The value must be between 0 and 25. The default is 1, which indicates a single attempt to reconnect. A value of 0 means that PROC OLAP will not attempt to reconnect.

MAX_RETRY_WAIT=
specifies the maximum number of seconds to wait before PROC OLAP attempts to reconnect to the metadata server. The time to wait (initially MIN_RETRY_WAIT) will be doubled each time a reconnect fails until this limit is reached. The value must be between 1 and 3600. The default value is the greater of MIN_RETRY_WAIT and 30.

MAXTHREADS=*n*
specifies the maximum number of threads that are used to asynchronously create the aggregation indexes. The processing engine calculates how many threads are needed based on the number of indexes that are being created and the INDEXSORTSIZE= value. This option sets a limit on the number of threads regardless of the number that is calculated by the processing engine. However, if the processing engine determines that fewer than the maximum number of threads is needed, then only the calculated number of threads are used.

The default value is the value of the SAS system option SPDEMAXTHREADS or 0. If the value is 0, then the processing engine determines the number of threads based on the number of indexes that are created plus the available memory. The maximum value is 65,536 threads.

MIN_RETRY_WAIT=
specifies the initial number of seconds to wait before PROC OLAP attempts to reconnect to the metadata server. This delay applies to a second attempt to reconnect to the metadata server. There is no delay for the first attempt to reconnect. The value must be between 1 and 3600 (one hour). The default value is the lesser of MAX_RETRY_WAIT and 30.

MLSCAPUPD
updates the captions and descriptions for multilingual cubes that were originally built with the CUBETABLECAPPREF and/or DIMTABLECAPPREF options. MLSCAPUPD must be used in conjunction with UPDATE_DISPLAY_NAMES. When the MLSCAPUPD and UPDATE_DISPLAY_NAMES options are specified together, the caption data sets are reread and the multilingual captions and descriptions are updated.

MLSID=*n*
a positive integer identifier between 0 and MACINT (2147483647) that identifies the observation in the data set that contains the translated caption and description for the cube. This identifier is expected in the MLSID column of the data set specified by CUBETABLECAPPREF=, CUBETABLELIBREF=, and the USER_DEFINED_TRANSLATIONS statement.

NO_NWAY
prevents PROC OLAP from automatically creating an NWAY aggregation (the crossing of all dimension levels) for the new cube. The automatically created NWAY is usually the largest in the cube and most resembles the content of the unsummarized data source. If you use this option, then the input data source that is specified with the DATA= or FACT= option must be available at run time. Otherwise, queries that are not covered by other aggregations will fail.

NONUPDATEABLE
specifies that the dimension (or dimensions) for a cube should be built with the minimum amount of disk space to represent the members available when the cube is created. By default, new dimensions are built to allow for new members to be added in future updates. NONUPDATEABLE is valid only when the cube is first created.

If this option is set, no new members can be added to the dimension in future updates of the cube. This option can be specified either in the PROC OLAP statement or on the individual DIMENSION statements. To make individual dimensions non-updateable, use this option in the DIMENSION statement instead.

OUTCUBE=*cube-name2* | *folder-path/cube-name* <*(cube)*>
specifies a valid SAS name for the new cube registration. For naming guidelines, see “Naming Guidelines and Rules for the SAS OLAP Server ” on page 51 . The

folder path that is specified is a metadata folder path. It exists only in the metadata. The OUTCUBE= option can be used in combination with either RENAME= option or the ADD_DATA option. If you are updating a cube, you can use OUTCUBE= option with the ADD_DATA option to specify the location of the new cube registration. If you are renaming an existing cube, you can use the OUTCUBE= with the RENAME option. OUTCUBE= does not create a copy of existing cubes.

In SAS 9.4, cubes are identified both by their location in an OLAP schema and by their association with a metadata folder. By default, if no folder-path is given, the cube will be associated with the folder of the original cube's OLAP schema, or, if specified, with the folder of the OLAP schema specified in the OUTSCHEMA= option. When you specify a metadata folder, you can add the metadata type (CUBE) after the cube name. The cube name must be unique within the OLAP schema and the metadata folder. The specified metadata folder path must exist.

Here are some examples of the OUTCUBE= option:

- Example 1

```
proc olap rename cube=MYCUBE_A outcube='/Shared Data/Cubes/MYCUBE_B';
metasvr ... olap_schema='SASApp - OLAP Schema';
run;
```

This example creates a cube registration named MYCUBE_B in the schema **SASApp - OLAP Schema** and the folder **Shared Data/Cubes**.

- Example 2

```
proc olap add_data cube=MYCUBE_A outcube='MYCUBE_B'
/* more options */; metasvr ... olap_schema='SASApp - OLAP Schema';
/* more statements *
/ run;
```

This example creates a registration named MYCUBE_B for the updated version of the cube MYCUBE_A. The new registration is located in the same OLAP schema and folder as MYCUBE_A.

OUTSCHEMA=*olap-schema-name*
specifies the name of the OLAP schema into which a new cube should be placed. The OLAP schema must already exist within the SAS Metadata Server and the metadata (or a repository on which it depends) specified in the METASVR statement.

You can use OUTSCHEMA= in combination with the RENAME option to move an existing cube registration into a different OLAP schema. Changing the OLAP schema for the cube is strictly a change to the metadata and does not result in a physical move. You can simultaneously use the OUTCUBE= option to also change the cube's name. In addition, OUTSCHEMA= can be used in combination with the ADD_DATA option to specify the target OLAP schema for the registration of the updated cube.

PARTSIZE=*partition-size*
specifies the partition size in megabytes of the aggregation table partitions and their corresponding index components. The default value is 128 megabytes. The minimum value is 16 megabytes.

Note: This option applies to the automatically created NWAY and all aggregations that do not explicitly specify a PARTSIZE= option in the AGGREGATION statement.

PATH='*pathname*'
specifies the physical path to the location of a new cube. Within the specified path, the cube is stored in a directory that uses the name of the cube. However, if the cube folder already exists, a unique subdirectory is generated. When a cube is renamed with the OUTCUBE= option, either with the RENAME or ADD_DATA option, the physical pathname does not change. Here is an example where this might cause a unique subdirectory to be generated:

- Create a cube named MRKTDATA.
- Rename the cube MRKTDATA as NEW_MARKET_DATA (or update MRKTDATA and specify OUTCUBE=NEW_MARKET_DATA).
- Create another cube named MRKTDATA.

The new cube MRKTDATA would have a cube folder of `C:\v9cubes\MRKTDATA1` because the cube folder `C:\v9cubes\MRKTDATA` still exists for cube NEW_MARKET_DATA.

REGISTER_ONLY
specifies that metadata for a cube is to be registered, but the cube is not to be physically built. All of the metadata for the cube is added to the SAS Metadata Repository. The physical cube can be built later using the existing metadata registration, with either the short form of PROC OLAP or in SAS OLAP Cube Studio. Note that all data sets must physically exist at registration time. The data sets can be empty; they do not need to contain data. Complete data sets are required when the cube is physically built.

RENAME
indicates that the cube should be renamed. Renaming a cube updates the metadata for the cube but does not change the file structure or physical location of the cube. This process requires an exclusive lock on the cube. If the cube is being queried by SAS OLAP Server sessions, the cube will need to be disabled (see the OLAPOPERATE procedure) before it is renamed.

Here is the syntax usage for the RENAME option:

PROC OLAP RENAME CUBE=*cube-name1* <OUTCUBE=*cube-name2*> <OUTSCHEMA=*olap-schema-name*>; METASVR options; RUN;

REORGANIZE_LEVELS | REORG_LEVELS
recognized as an advanced option that enables you to reorganize all eligible levels for a cube. Use this option only when directed to do so by the SAS log, after a failed cube update. The levels that are eligible for reorganization are those that were previously updated with new captions.

SECURITY_SUBSET= |
controls how permission conditions are interpreted at query time. With SECURITY_SUBSET= YES, cell values are recalculated at query time based on the security subset defined by the active permission conditions for the given user. SECURITY_SUBSET=NO does not recalculate the cell values. The default value (NO) includes all members within a total.

Note: This option can also be found in SAS OLAP Cube Studio. See the option **Include secured member values in presummarized computations**on the Cube Designer - General page of the Cube Designer wizard.

SEGSIZE=*rows-per-segment*
specifies the number of observations (table rows) in the file segment of the index component. The value is expressed in multiples of 1,024. The minimum value is 1 (1,024 rows). The segmented indexes are used to optimize the processing of

WHERE expressions. Each parallel thread is given a segment of the table to evaluate that is equal to the value of the SEGSIZE= option multiplied by 1,024. The default value is 8 (8 x 1,024 = 8,192 rows).

Note: This option applies to the NWAY aggregation and all aggregations that do not explicitly specify a SEGSIZE= option in the AGGREGATION statement.

SORTSEQ= | < (*collation-options*)>
enables you to control how the level member data is sorted. With this option, you can override the default sort order on a cube that contains data in a single language. For example, if your have German data, you might prefer to sort your level members with the PHONEBOOK collation rule, as shown in the following abbreviated PROC OLAP statement:

```
PROC OLAP cube=mycube SORTSEQ=LINGUISTIC (COLLATION=PHONEBOOK LOCALE=de_DE);
```

In German (de_DE above), the PHONEBOOK collation rule sorts differently than the default dictionary sort. Vowels with umlauts are sorted with and just above similar vowels without umlauts. The sort results change as follows:

```
Dictionary sort:     PHONEBOOK sort:

1 Mader Ernst              1 Mader Ernst
2 Mader Fritz              2 Mader Fritz
3 Mader Josef              3 Mader Josef
4 Meder Bruno              4 Meder Bruno
5 Meder Regina             5 Meder Regina
6 Meier Hans               6 Meier Hans
7 Mlynek Mike              7 Mlynek Mike
8 Molitor Martina          8 Möller Ellen       *
9 Möller Ellen             9 Möller Georg       *
10 Möller Georg            10 Möller Sabine     *
11 Möller Sabine           11 Molitor Martina   *
12 Mras Cindy              12 Mras Cindy
13 Muller George           13 Müller Christina  *
14 Muller Pam              14 Müller Heinz      *
15 Muller Susan            15 Müller Max        *
16 Müller Christina        16 Müller Renate     *
17 Müller Heinz            17 Muller George     *
18 Müller Max              18 Muller Pam        *
19 Müller Renate           19 Muller Susan      *
20 Mwamba Ed               20 Mwamba Ed
21 Myrzik Peter            21 Myrzik Peter
22 Mzyk Mary               22 Mzyk Mary

* indicates names that changed position in the PHONEBOOK sort.
```

Note: The MDX ORDER() function honors the collation sequence that was selected when the cube was built.

LINGUISTIC	sorts characters according to the Unicode Collation Algorithm and a specified locale. For more information on the algorithm, see http://www.unicode.org.
UCA	is a synonym of LINGUISTIC.

collation-options specifies in parenthesis values for the COLLATION option and/or the LOCALE option.

COLLATION=*collation-rules* modifies the default UCA sort algorithm by applying one or more comma-separated collation rules from the following table. The default sort is based on the dictionary order of the specified locale.

LOCALE=*posix-locale-name* specifies a locale name other than the default locale, using 5-character Posix notation. As an example of Posix notation, the locale name es_AR identifies the Spanish language as it is used in Argentina.

To see a complete list of the Posix locale names that are supported by SAS, refer to the topic entitled “LOCALE= Values and Default Settings for ENCODING, PAPERSIZE, DFLANG, and DATESTYLE Options,” in the *SAS 9.4 National Language Support (NLS): Reference Guide*.

The following table lists the available collation rules, the names of which can be specified as the value of the COLLATION option.

Table 12.3 *Collation Rules Available as Values for the COLLATION option.*

Names of Collation Rules	Description
BIG5HAN	Specifies Pinyin ordering for Latin and specifies big5 charset ordering for Chinese, Japanese, and Korean characters.
DIRECT	Specifies a Hindi variant.
GB2312HAN	Specifies Pinyin ordering for Latin and specifies gb2312han charset ordering for Chinese, Japanese, and Korean characters.
PHONEBOOK	Specifies a telephone-book style for ordering of characters. Select PHONEBOOK only with the German language.
PINYIN	Specifies an ordering for Chinese, Japanese, and Korean characters based on character-by-character transliteration into Pinyin. This ordering is typically used with simplified Chinese.
POSIX	Is the Portable Operating System Interface. This option specifies a "C" locale ordering of characters.
STROKE	Specifies a nonalphabetic writing style ordering of characters. Select STROKE with Chinese, Japanese, Korean, or

Names of Collation Rules	Description
	Vietnamese languages. This ordering is typically used with Traditional Chinese.
TRADITIONAL	Specifies a traditional style for ordering of characters. For example, select TRADITIONAL with the Spanish language.

Note: Specifying the LINGUISTIC option without specifying COLLATION= will result in the default collation sequence for that locale.

SYNCHRONIZE_AGGRS | SYNC_AGGRS
indicates that the cube aggregations will be checked for consistency with the current members trees for all shared dimensions that it uses. If the cube requires synchronization, the cube aggregations are rebuilt. This option would be necessary after a shared dimension used by the cube has been reorganized (see REORGANIZE_LEVELS).

SYNCHRONIZE_COLUMNS
synchronizes the column names stored in the cube's internal metadata with the input column names for a cube. This is necessary when data set column names have changed since the cube was last built. This applies to cube levels, measures, and properties.

Note: If column names for a cube have changed and the cube was not synchronized, you will not be able to update the cube. If the cube has presummarized aggregations, you will not be able to query the cube.

UPDATE_DISPLAY_NAMES
enables you to update the captions for dimensions, hierarchies, levels, member properties, and measures. This option specifies that captions or descriptions on the cube are going to be modified.

The UPDATE_DISPLAY_NAMES option is allowed with the ADD_DATA option or by itself in the PROC OLAP statement. It can also be used to change the description or caption on the DIMENSION, HIERARCHY, LEVEL, MEASURE, and PROPERTY statements.

Here is the syntax usage for the UPDATE_DISPLAY_NAMES option:

```
PROC OLAP CUBE=cubename DESCRIPTION="new description"
            UPDATE_DISPLAY_NAMES DT=newdtname; METASVR options;
        DIMENSION TIME DESCRIPTION="New TimeDescription"
            CAPTION="New TimeCaptions";
```

It is also possible to remove a drill-through table from a cube by specifying the DT_TABLE= option with the UPDATE_DISPLAY_NAMES option. By specifying DT_TABLE=_NULL_ UPDATE_DISPLAY_NAMES, the current drill-through table is cleared from the cube file and the metadata association is removed.

Here is the syntax usage for removing a drill-through table:

```
PROC OLAP CUBE=cubename DT_TABLE=_NULL_ UPDATE_DISPLAY_NAMES;
METASVR <Connection options>;
```

```
RUN;
```

Specific considerations for the UPDATE_DISPLAY_NAMES option include the following:

Drill-through tables	These can be updated with the UPDATE_DISPLAY_NAMES option. When you rename a cube, only the cube description or drill-through table name can be updated via the UPDATE_DISPLAY_NAMES option. You can remove a drill-through table with the DT_TABLE=_NULL_ UPDATE_DISPLAY_NAMES syntax.
NUNIQUE measures	When the UPDATE_DISPLAY_NAMES option is specified, the NUNIQUE measure's INCLUDE_CALCULATED_MEMBERS option can be toggled ON or OFF (with the addition of the NOINCLUDE_CALCULATED_MEMBERS option).
RENAME	The only description that can change during a RENAME of a cube is the cube description (the drill-through table name can also be changed during a RENAME). Other descriptions and captions must be altered in a separate step after the RENAME is executed.

Note: To update captions in multilingual cubes, you must specify the MLSCAPUPD option along with the UPDATE_DISPLAY_NAMES option.

UPDATE_IN_PLACE
enables you to control how the updated cube is made available to users. An update of the existing cube will occur. UPDATE_IN_PLACE can be used only when the ADD_DATA option is specified. It cannot be used in combination with the OUTCUBE= or OUTSCHEMA= options.

WORKPATH=(*'pathname1' ...'pathnameN'*)
specifies one or more locations for temporary work files.

For all operating environments except z/OS and VMS, if the WORKPATH= option is not specified, PROC OLAP uses the SPDEUTILLOC= system option. If SPDEUTILLOC= is not specified, PROC OLAP uses the UTILLOC= system option. If UTILLOC= is not specified, or if you do not have Write access to the specified path, the following message is generated:

```
ERROR: Cannot create temporary index for proc olap.
NOTE: The SAS System stopped processing this step
because of errors.
```

In the z/OS operating environment, PROC OLAP uses the SPDEUTILLOC= system option only.

METASVR Statement

Identifies the SAS metadata repository in which existing cube metadata information exists or in which metadata about a new cube is stored. The METASVR statement options can be used to override the metadata repository connection values that are specified through SAS start-up options.

Syntax

METASVR OLAP_SCHEMA='*schema-name*' <*options*>;

Details

Requirements

OLAP_SCHEMA='*schema-name*'
is a string that specifies the name of the schema that has been defined in a SAS metadata repository. The name can be a maximum of 60 characters. The OLAP schema specifies which group of cubes that a SAS OLAP Server can access. Each OLAP schema can be accessed by multiple SAS OLAP Servers. However, each SAS OLAP Server has access to only one OLAP schema. When using embedded blanks or special characters in the schema name, enclose the name in quotation marks.

Options

HOST='metadata-server-host-name'
is a string that specifies the IP address of the metadata repository host. An example is 'misdept.us.mar.com'. The address can be a maximum of 256 characters. When using lowercase letters, embedded blanks, or special characters in the host name, enclose the name in quotation marks.

PORT=*port-number*
specifies the numeric value of the port on which the metadata repository resides.

PW=*'password'*
is a string that specifies the password for the user identified with the USERID= option. The password can be a maximum of 512 characters. When using lowercase letters, embedded blanks, or special characters in the password, enclose the password in quotation marks.

USERID=*'userid'*
is a string that specifies the user's identification for the specified metadata repository. The identification can be a maximum of 256 characters. When using lowercase letters, embedded blanks, or special characters in the user ID, enclose the user ID in quotation marks.

Note: During an interactive SAS session, if connection information is not available either through start-up settings or through a METASVR statement, then you are prompted for the missing information. For more information about SAS start-up options, see the *SAS Language Reference: Dictionary*.

Example

Here is an example of a METASVR statement with all of its options set:

```
metasvr olap_schema='Banking Schema'
host='misdept.us.mar.com'
port=9999
userid=jjones
pw='my password';
```

DIMENSION Statement

Defines the logical and hierarchical relationships between the variables in the input data.

Syntax

DIMENSION *dim-name* HIERARCHIES=(*hier–nam ... hier-nameN*) <*options*>;

Details

Overview

At least one DIMENSION statement must be specified when the cube is created. The maximum number of dimensions that can be defined in a cube is determined by combining the number of dimensions with the number of multiple hierarchies that are defined in those dimensions. The maximum value of that sum is 128. Mathematically, the sum is expressed as follows:

```
MaxDims = NumDims + NumMultipleHierarchies = 128
```

All hierarchies other than the first hierarchy in each dimension apply to the total.

- 128 dimensions, each dimension has 1 hierarchy
- 127 dimensions, 1 dimension has 2 hierarchies
- 126 dimensions, 1 dimension has 3 hierarchies
- 126 dimensions, 2 dimensions have 2 hierarchies

A DIMENSION statement must include the name of at least one hierarchy in its HIERARCHIES= option. In addition, a HIERARCHY statement must include the name of at least one level in its LEVELS= option. Note that you cannot use the same level in more than one dimension.

You can use LEVEL statements to specify a time period for each level in a TIME dimension. You can also use LEVEL statements to supply information such as a level-specific sort order or a level description.

Required Arguments

***dim-name* | *folder-path/dim-name* <(SHARED DIMENSION)>)**

dim-name names a dimension by using a valid SAS name up to 32 characters in length. Shared dimensions use the longer form, with the path, name, and

keyword. For naming guidelines, see “Naming Guidelines and Rules for the SAS OLAP Server ” on page 51. For information on shared dimensions, see “Understanding Shared Dimensions” on page 368, and “Building a Shared Dimension” on page 285.

HIERARCHIES=*(hier-name...hier-nameN)*
specifies the name of one or more hierarchies as defined by HIERARCHY statements.

Options

CAPTION=*'string'*
specifies a maximum of 200 characters that can be used to create a meaningful description of the dimension. Third-party applications that report on cube data might display this description. If the text includes blank spaces or any characters that are not permitted in a valid SAS name, then enclose the text within quotation marks. The default value is the name of the dimension, as specified by the required argument *dim-name*.

Note: Cubes that are built with captions that are longer than 200 characters cannot be fully registered in the SAS Metadata Repository. Captions will be truncated to 200 characters.

DESC | DESCRIPTION=*'string'*
specifies any number of characters that can be used to create a meaningful description of the dimension. Third-party applications that report on cube data might display this description. If the text includes blank spaces or any characters that are not permitted in a valid SAS name, then enclose the text within quotation marks. The default value is the value of the CAPTION= option.

DIMKEY=*dimension-table-column*
specifies the name of a column in the dimension table that is specified in the DIMTBL= option. That column must contain values that correspond to fact key values in the fact table and be a value that corresponds to a unique combination of level values in the fact table.

Note: The corresponding fact key is specified with the FACTKEY= option. The fact table is specified with the FACT= option in the PROC OLAP statement.

For example, for a dimension that consists of three levels—COUNTY, REGION, and STATE—a dimension key named AREA_ID might exist. In this dimension, each unique value of AREA_ID corresponds to a unique combination of COUNTRY, REGION, and STATE.

Table 12.4 *Sample Dimension Data That Illustrates How Unique DIMKEY Values Correspond to Unique Combinations of Level Values*

AREA_ID	COUNTRY	REGION	STATE
1	USA	East	NY
2	USA	East	NJ
3	USA	West	CA

AREA_ID	COUNTRY	REGION	STATE
4	USA	West	AZ
5	CANADA	East	QUEBEC
6	CANADA	West	BRITISH COLUMBIA

DIMTBL=*libname.memname*
specifies the two-level SAS name or a dimension table that matches the fact table that is specified with the FACT= option in the PROC OLAP statement. The dimension table must contain one column for each dimension level and each level property and one column for the dimension key. However, if the dimension key is also a level, then the dimension table needs to have only as many columns as there are levels in the dimension. Member metadata for the dimension is derived from the information in the level columns of the dimension table.

You can also specify data set options with DIMTBL=. Options are stored within the cube and reapplied when the cube is unpacked or rebuilt. For more information, see “Data Set Options” in *SAS Language Reference: Concepts*.

Note: The same dimension tables can be used to load cubes that have some, but not all, dimensions in common. This means that it is possible for multiple cubes to share the same dimension data.

Note: If you are building a cube that will contain multiple national languages, then replace the DIMTBL= option with DIMTABLELIBREF= and DIMTABLEMEMPREF= options. In addition, you must create a USER_DEFINED_TRANSLATIONS statement.

DIMTABLECAPPREF=*caption-table-prefix*
specifies the member prefix for the translated dimension CAPTION tables. The member prefix is the prefix of the data set name. The suffix of the name is provided by the USER_DEFINED_TRANSLATIONS statement. This specification is optional but if used must differ from the DIMTABLEMEMPREF option. For example, if the caption member prefix is `dealdimcap_` and the suffix is `da_DK`, then PROC OLAP looks for a data set named `dealdimcap_da_DK.sas7bdat` in the library that is specified by the DIMTABLELIBREF= option. This option is used in conjunction with the DIMTABLELEBREF=option and the USER_DEFINED_TRANSLATIONS statement.

If this option is not used, then captions appear in the default language.

DIMTABLELIBREF=*mls-library*
specifies the library for the data sets that exist, for this dimension, in each language that is specified by the USER_DEFINED_TRANSLATIONS statement. The library is associated with the dimension and not the language. If you choose to put your data sets in different librefs based on languages, you must use a concatenated libref so that it appears to the PROC as a single libref.

This option is required if you are using the Multiple Language Support capabilities of the SAS OLAP Server. This option is used in conjunction with the DIMTABLEMEMPREF= option.

Note: If you are building a cube that will contain multiple national languages, then DIMTABLELIBREF= and DIMTABLEMEMPREF= are required instead of DIMTBL=.

DIMTABLEMEMPREF=
specifies the member prefix for the translated dimension tables. The member prefix is the prefix of the data set name. The suffix of the name is provided by the USER_DEFINED_TRANSLATIONS statement. For example, if the member prefix is **`dealdim_`** and the suffix is **`da_DK`**, then PROC OLAP looks for a data set named **`dealdim_da_DK.sas7bdat`** in the library that is specified by the DIMTABLELIBREF= option. DIMTABLEMEMPREF= is required if you are using the Multiple Language Support capabilities of the SAS OLAP Server. It is used in conjunction with the DIMTABLELIBREF= option and the USER_DEFINED_TRANSLATIONS statement.

Note: If you are building a cube that will contain multiple national languages, then DIMTABLELIBREF= and DIMTABLEMEMPREF= are required instead of DIMTBL=.

EMPTY_CHAR='*string*'
specifies the text string that identifies members of character levels that are to be skipped or disregarded. Members are skipped in order to create ragged or unbalanced hierarchies, as described in "Defining Cube Hierarchies" on page 69.

To be skipped, a member in a character level must have a caption whose value matches the value of the EMPTY_CHAR= option. For example, if a member in a character level is skipped, and if the caption of that member is **`Empty`**, then the EMPTY_CHAR= option is specified as follows: `empty_char='Empty'`

The maximum length of the quoted string is 256 characters.

When specified in the HIERARCHY statement, the EMPTY_CHAR= option overrides (for that hierarchy) any specification of the EMPTY_CHAR= option in the PROC OLAP statement. In turn, the EMPTY_CHAR= option in the HIERARCHY statement is overridden by the EMPTY= or IGNORE_EMPTY options in the LEVEL statements in that hierarchy.

To skip members in numeric levels, use the EMPTY_NUM= option.

EMPTY_NUM='*string*'
specifies the text string that identifies members of numeric levels that are to be skipped or disregarded. Members are skipped in order to create ragged or unbalanced hierarchies, as described in "Defining Cube Hierarchies" on page 69.

To be skipped, a member in a numeric level must have a caption whose value matches the value of the EMPTY_NUM= option. For example, if a member in a numeric level is skipped, and if the caption of that member is **`Empty`**, then the EMPTY_NUM= option is specified as follows: `empty_num='Empty'`

The maximum length of the quoted string is 256 characters.

When specified in the HIERARCHY statement, the EMPTY_NUM= option overrides (for that hierarchy) any specification of the EMPTY_NUM= option in the PROC OLAP statement. In turn, the EMPTY_NUM= option in the

HIERARCHY statement is overridden by the EMPTY= or IGNORE_EMPTY options in the LEVEL statements in that hierarchy.

Note: If there is no format that is associated with the member value, then BEST12 is used as the format.

To skip members in character levels, use the EMPTY_CHAR= option.

FACTKEY=*fact-table-column*
specifies the name of the column in the fact table that corresponds to the dimension table column that is specified with the DIMKEY= option. The name does not have to match the DIMKEY name. Referring back to the previously discussed example, the FACTKEY name could be CUST_NO even though the DIMKEY name is CUSTOMER_ID. However, even if the names are different, the underlying data must match. For example, you must match numeric columns with numeric columns and character columns with character columns. In addition, if the FACTKEY is a character column, then it must be the same length as the DIMKEY column. If the FACTKEY is a numeric column, then it is handled as a decimal precision number (rather than as an integer).

Restriction For shared dimensions, the FACTKEY= option is not allowed. Instead, the fact key is specified with the USE_DIMENSION statement.

FORCE
enables deletion of existing shared dimension during a shared dimension build without having to use a DELETE option in a separate invocation.

If the shared dimension already exists in the specified schema, then the shared dimension will be deleted prior to building the new shared dimension. If this is a short-form build, using code generated by SAS OLAP Cube Studio, then this is equivalent to a DELETE_PHYSICAL followed by the build. If a metadata folder path is specified (for example DIMENSION <dim-name> SHAREDFORCE="/Shared Data/SD/MySD" FORCE), then the shared dimension, if it exists, must exist in the specified folder or the deletion will fail, as shown in the following example:

```
DIMENSION "/Shared Data/SD/MySD"  SHARED FORCE
```

This is how DELETE and DELETE_PHYSICAL behave currently.

Restriction The FORCE option on the DIMENSION statement is for shared dimensions only. The FORCE option is also valid for cubes when used on the PROC OLAP statement with CUBE=.

CAUTION **The FORCE option does not include a rollback feature.** If the shared dimension build fails, there is no rollback of the original shared dimension.

IGNORE_EMPTY
specifies that, for this hierarchy, any values that were specified for the EMPTY_CHAR= and EMPTY_NUM= options in the PROC OLAP statement are to be ignored. This option can be overridden by specifications of EMPTY_CHAR= and EMPTY_NUM= in the same HIERARCHY statement. The IGNORE_EMPTY option can also be overridden in subsequent LEVEL statements by using the EMPTY= option. For further information, see “Defining Cube Hierarchies” on page 69.

MAP_SERVICE=*MapServiceName*
specifies the map service to which this cube should be linked. The dimension must have a dimension type of GEO. The map service must be a map service defined with the Esri server specified previously. There can be only one map service per GEO dimension.

MLSID=*n*
is a positive integer identifier between 0 and MACINT (2147483647) that identifies the observation in the data set that contains the translated caption and description for the dimension. This identifier is expected in the MLSID column of the data set specified by DIMTABLECAPPREF=, DIMTABLELIBREF=, and the USER_DEFINED_TRANSLATIONS statement.

Note: For further information about MLS caption tables and MLSID, see "USER_DEFINED_TRANSLATIONS Statement" on page 361.

NONUPDATEABLE
specifies that the dimension should be built with the minimum amount of disk space to represent the members available when the cube is created. By default, new dimensions are built to allow for new members to be added in future updates. NONUPDATEABLE is valid only when the cube is first created.

If this option is set, no new members can be added to the dimension in future updates of the cube. If you want to make all dimensions of a cube non-updateable, you can use the NONUPDATEABLE option in the PROC OLAP statement instead.

PATH=*'path-name'*
specifies the physical or logical path to the location of a new shared dimension. Within the specified path, the dimension is stored in a directory that uses the name of the dimension. For example, if you enter the path `'C:\shared_dimensions\Customer'`, then you should enclose the path within quotation marks.

SHARED
specifies that the dimension that is being referenced is a shared dimension.

SORT_ORDER= | | | |
specifies a sort order for all levels in the dimension. Values that are returned from queries appear in this order by default. This setting is overridden if sort order is set in a LEVEL statement.

Default: *ASCENDING*

To specify a sort order for each level within a dimension, set the SORT_ORDER= option in each LEVEL statement. Values that are returned from queries appear in this order. The sort order can be changed at query time using the MDX ORDER functions.

TYPE= |
identifies the dimension as a TIME or GEO dimension. The GEO type is used when defining Esri map information for a cube. There can be only one GEO and one TIME dimension for a cube. You must set this option for a TIME or GEO dimension. TIME and GEO are the only valid values for this option. You can use LEVEL statements to specify the time period of each level in the TIME dimension. Specifying TYPE=TIME also enables you to use the MDX time series functions during data query.

Note: In order to add geographic information to an existing cube, you must use SAS OLAP Cube Studio. The OLAP procedure does not support adding the GEO type to an existing dimension.

UPDATE_DIMENSION
enables you to add new members to a cube and update member properties for existing members of the dimension. UPDATE_DIMENSION can be used only on a dimension without the NONUPDATEABLE designation. It can be used with or without the ADD_DATA option in the PROC statement. It can be used with one of the following parameters:

MEMBERS	The dimension table currently associated with the dimension should be read and processed for new members of every hierarchy in that dimension. If the ADD_DATA option is used, this is the default behavior for cubes loaded from a detail table or a star schema. If the ADD_DATA option is not used, this parameter can be applied to individual dimensions for cubes loaded from star schemas only. It is not valid for a cube loaded from a detail table. Thus this option provides a way to update individual dimensions without adding data to the cube.
MEMBERS_AND_PROPERTIES	The dimension table currently associated with the dimension should be read and processed for new members and member properties for existing members should be changed with the new values in the dimension table. If the ADD_DATA option is used, this option can be used for cubes loaded from a detail table or a star schema. If the ADD_DATA option is not used, it can be applied to individual dimensions for cubes loaded from star schemas only. It is not valid for a cube loaded from a detail table. Thus this option provides a way to update individual dimensions without adding data to the cube.
OFF	The dimension table currently associated with the dimension should not be read at all. When this parameter is used, it applies only to cubes loaded from a star schema and never to a cube loaded from a single detail table. If the ADD_DATA option is used on the PROC, this option must be used on any UPDATEABLE dimensions whose dimensions tables should not be processed, since the default behavior is UPDATE_DIMENSIONS= MEMBERS. If the ADD_DATA option is not used on the

PROC, this is the DEFAULT option for all dimensions in the cube.

USE_DIMENSION Statement

Specifies a shared dimension to be included in the cube.

Syntax

USE_DIMENSION [*dim-name* | *"/folder/dim-name"*]
[OLAP_SCHEMA=*schema-name*] FACTKEY=*fact-table column*;

Details

Required Arguments

dim-name | "/folder/dim-name"
provides the name of the dimension or the location and name of the dimension referenced.

OLAP_SCHEMA= *schema-name*
specifies the name of the OLAP schema. The OLAP schema option is required if the shared dimension's schema is different from the schema specified in the METASVR statement.

FACTKEY= *fact-table column*
provides the name of the key column in the fact table that corresponds to the DIMKEY key column in the dimension table.

Note: HIERARCHY, LEVEL, and PROPERTY statements are not needed or allowed for the shared dimension. The structure of the dimension was defined when the shared dimension was originally built.

Note: If the EMPTY_CHAR= or EMPTY_NUM= options are specified in the PROC OLAP statement, those values do not apply to the shared dimension. These options might be specified in the original DIMENSION statement for which this dimension is shared.

See Also

"Using a Shared Dimension in a Cube" on page 373

HIERARCHY Statement

Specifies the navigational order of the levels in a dimension.

Syntax

HIERARCHY *hier-name* LEVELS=(*level–name1 <level–name2 ...level-nameN>*) <*options*>;

Details

Overview

You must define at least one hierarchy for each dimension. Specifically, each DIMENSION statement must identify at least one unique HIERARCHY statement. The maximum number of hierarchies that can be defined in a cube is 128. Mathematically, the sum is expressed as follows:

```
MaxHiers = NumMultHiers + NumDimensions = 128
```

All hierarchies other than the first hierarchy in each dimension apply to the total. Levels in the same dimension can be shared between hierarchies. You can have a maximum of 19 levels per hierarchy. There is no limit to the number of hierarchies per dimension.

Here are some examples of cubes that meet the maximum number of hierarchies:

- 128 dimensions, each dimension has 1 hierarchy
- 127 dimensions, 1 dimension has 2 hierarchies
- 126 dimensions, 1 dimension has 3 hierarchies
- 126 dimensions, 2 dimensions have 2 hierarchies

Required Arguments

hier-name
specifies a valid SAS name for the hierarchy. This name is also used in the HIERARCHIES= option in the DIMENSION statement. The *hier-name* cannot be the same as any of its level names. Hierarchy names must be unique within the cube. If the hierarchy that you are defining is the only one in the dimension, then the hierarchy name must match the dimension name. For other naming guidelines, see “Naming Guidelines and Rules for the SAS OLAP Server ” on page 51.

LEVELS=(*level-name1 <level-name2 ...level-nameN>*)
specifies a valid SAS name for at least one level. These names correspond to the level names used in any optional LEVEL statements. Level names must be

unique within a cube and cannot be the same as the *hier-name*. (You can use a column as a level even if it is also being used as a measure.) Enter one or more names, separated by a space. Enter the level names in the order in which you want them to be used, beginning with the top level. For naming guidelines, see "Naming Guidelines and Rules for the SAS OLAP Server " on page 51.

If the hierarchy is part of a TIME dimension, then the levels must be listed in order from most general to least general based on their assigned TYPE. For example, a TYPE=YEAR level must be listed before a TYPE=QUARTER level. Level names should not conflict with the ALL member name in the HIERARCHY statement. You should not use *ALL* for the level name.

Options

ALL_MEMBER=*'string'*
specifies the caption for the ALL member of the hierarchy.

The ALL member name should not conflict with level names in the LEVEL or HIERARCHY statement. When selecting the ALL member name, follow these guidelines:

- The ALL member caption should not be `all`.
- A level name within the hierarchy should not be named `all`.
- The ALL member caption should not be a level name that is within the hierarchy `all`.

When a cube is built, a warning message is displayed in the SAS log and in SAS OLAP Cube Studio if a conflict between the ALL member name and a level name is detected.

CAPTION=*'string'*
specifies a maximum of 200 characters that can be used to create a meaningful description of the hierarchy. Third-party applications that report on cube data might display this description. If the text includes blank spaces or any characters that are not permitted in a valid SAS name, then enclose the text within quotation marks. The default value is the *hier-name*.

Note: Cubes that are built with captions that are longer than 200 characters cannot be fully registered in the SAS Metadata Repository. Captions will be truncated to 200 characters.

DEFAULT
identifies a hierarchy as the default hierarchy for the dimension that is defined by the DIMENSION statement. The default value is the first hierarchy listed for the dimension.

DESC | DESCRIPTION=*'string'*
specifies any number of characters that can be used to create a meaningful description of the hierarchy. Third-party applications that report on cube data might display this description. If the text includes blank spaces or any characters that are not permitted in a valid SAS name, then enclose the text within quotation marks. The default value is the hierarchy caption, which might be the default, *hier-name*.

DEFAULT_MEMBER= *'member-unique-name'*
specifies a default member when an MDX query opens a cube. This member is used to aggregate the hierarchy when no other criteria are provided. The

member name must be fully qualified, as in `"[TIME].[All TIME].[2010].[December]"`. When the cube is built, the ALL member is the default member. The default can be changed using this option along wit the UPDATE_DISPLAY_NAMES option.

The named member must be part of the members to which the user is permitted access. If the member name cannot be found, or if the member is inaccessible, then the ALL member is used as the default member.

The DEFAULT_MEMBER= option is used with the PROC OLAP statement option UPDATE_DISPLAY_NAMES.

Specifying a calculated member as the default member can degrade performance.

MLSID=*n*
is a positive integer identifier between 0 and MACINT (2147483647) that identifies the observation in the data set that contains the translated caption and description for the hierarchy. This identifier is expected in the MLSID column of the data set specified by DIMTABLECAPPREF=, DIMTABLELIBREF=, and the USER_DEFINED_TRANSLATIONS statement.

Note: For further information on MLS caption tables and MLSID, see "USER_DEFINED_TRANSLATIONS Statement" on page 361.

Example

Here is an example of a HIERARCHY statement that specifies three levels:

```
hierarchy Geography
   levels=(country region division);
```

LEVEL Statement

Provides additional information about a level specified with the LEVELS= option in a HIERARCHY statement, and enables you to set options for ragged hierarchies. Each LEVEL statement must correspond to a HIERARCHY statement. You cannot have a level that does not belong to a hierarchy.

Syntax

LEVEL *level-name* <*options*>;

Details

Overview

For TIME dimensions, you can use LEVEL statements to specify a time period for each level in the dimension. However, if you specify the time period for one level, then you must specify the time period for all levels. You also use LEVEL statements to supply information such as a level description or a level-specific sort order. You can have a maximum of 256 levels per cube and a maximum of 19 levels per hierarchy.

Levels that are shared between hierarchies share the values of the options EMPTY_CHAR=, EMPTY_NUM=, EMPTY=, and IGNORE_EMPTY. These options are used to create ragged or unbalanced hierarchies, as described in “Defining Cube Hierarchies” on page 69.

When you rebuild a cube that has been physically deleted, the rebuilt cube still uses the formats that were originally used to build the cube and were saved in the cube's metadata. This means that the rebuilt cube does not automatically include any formatting changes that you might have made in the input data source. To manually specify the new formats, edit and rebuild the cube by using SAS OLAP Cube Studio.

Required Arguments

level-name

specifies a valid SAS name. For naming guidelines, see “Naming Guidelines and Rules for the SAS OLAP Server ” on page 51. This is the same name that is used in the LEVELS= option in the HIERARCHY statement. Level names must be unique within a cube.

Do not use `ALL` or `ALL` *`hiername`* as the level name, to avoid ambiguities during queries. The cube will build with such level names, but the SAS log will contain a warning.

Options

CAPTION=*'string'*

specifies a maximum of 200 characters that can be used to create a meaningful description of the level. Third-party applications that report on cube data might display this description. If the text includes blank spaces or special characters that are not permitted in a valid SAS name, then enclose the caption within quotation marks.

Note: Cubes that are built with captions that are longer than 200 characters cannot be fully registered in the SAS Metadata Repository. Captions will be truncated to 200 characters.

The default setting is the column's label in the input data source. If there is no label available, the default is the level name.

COLUMN=*column-name*
specifies the name of a column from the input data source. Use this option if the column name is not the same as the level name. This option is useful in the following scenarios:

- You want your level name to be different from the input column name.
- Different dimension tables for the cube each have a column with the same name.
- You want to use the same dimension table and columns in more than one dimension of your cube.
- You load multiple levels from the same input column (in combination with the FORMAT= option).

You can use a column as a level even if it is also being used as a measure.

DESC | DESCRIPTION=*'string'*
specifies any number of characters that can be used to create a meaningful description of the level. Third-party applications that report on cube data might display this description. If the text includes blank spaces or any characters that are not permitted in a valid SAS name, then enclose the text within quotation marks.

The default setting is the value of the CAPTION= option if one exists; otherwise, it is the column's label. If there is no label available, the default is the level name.

EMPTY='*string*'
specifies the text string that identifies members that are to be skipped or disregarded. Members are skipped in order to create ragged or unbalanced hierarchies, as described in "Defining Cube Hierarchies" on page 69.

To be skipped, a member must have a caption whose value matches the value of the EMPTY= option. For example, if a member is skipped, and if the caption of that member is **Empty**, then the EMPTY= option is specified as follows:

```
empty='Empty'
```

The maximum length of the quoted string is 256 characters.

The EMPTY= option overrides for that level any specification of EMPTY_CHAR=, EMPTY_NUM=, or IGNORE_EMPTY that might have been specified in the respective HIERARCHY or PROC OLAP statement.

ESRI_MAP_LAYER=*MapLayerName*
specifies the Esri map layer which should be associated with this level. It must be a map layer defined in the specified map service. The level must be a level contained in the dimension with dimension type GEO. Only one map layer can be defined per level. The same map layer can be specified for different levels in the dimension. However, the map layer is generally specified with a different map field indicated.

FORMAT=*sas-format-name*
specifies the SAS format to be used when reading in the member caption data from the input data source for the level. By default, the column format specified in the input table is used. However, you can override that format by using the SAS FORMAT statement in the PROC OLAP step or the FORMAT= option in the LEVEL statement.

This option can be particularly useful if you want to load multiple levels from the same column. A typical use is loading the levels of a TIME dimension. For example, if your input data contain a column with a SAS date values, you could load the TIME dimension levels, as shown in the following example.

```
dimension Time  hierarchies=(Time)  type=time ;
hierarchy Time  levels=(Year Quarter Month Day) ;
level Year  type=year  column=date_id  format=year. ;
level Quarter  type=quarters  column=date_id  format=qtr. ;
level Month  type=months  column=date_id  format=monname. ;
level Day  type=days  column=date_id  format=weekdate. ;
```

Note: Formats are only supported on numeric levels when building multilingual cubes. Character based formats are ignored if specified.

IGNORE_EMPTY
specifies that any value of the EMPTY_CHAR= option (for character levels) or EMPTY_NUM= option (for numeric levels) that was specified in the respective HIERARCHY or PROC OLAP statement is to be ignored. The level is not to be skipped in the cube build. For further information, see “Defining Cube Hierarchies” on page 69.

MLSID=*n*
is a positive integer identifier between 0 and MACINT (2147483647) that identifies the observation in the data set that contains the translated caption and description for the level. This identifier is expected in the MLSID column of the data set specified by DIMTABLECAPPREF=, DIMTABLELIBREF=, and the USER_DEFINED_TRANSLATIONS statement.

Note: For further information on MLS caption tables and MLSID, see “USER_DEFINED_TRANSLATIONS Statement” on page 361.

SORT_ORDER= | | | |
specifies a sort order for a level within a dimension. Values that are returned from queries appear in this order.

If a sort order is not specified in the DIMENSION statement or in the LEVEL statement, then the default order of ASCENDING is applied.

This setting overrides the SORT_ORDER= setting in the DIMENSION statement.

TYPE= | | | | | | | |
specifies the time period for the dimension levels when you specify the TYPE=TIME option in the DIMENSION statement.

If you specify a time period for one level in the TIME dimension, then you must specify the time period for all levels in the dimension. With regard to drill path, identify the levels from the most general time period to the most specific.

PROPERTY Statement

The PROPERTY statement assigns properties to specific levels within specified hierarchies. Each level can have more than one property assigned to it by using multiple PROPERTY statements. Property names must match the name of a column in the input data source, or you must use the COLUMN= option to specify the column name.

Syntax

PROPERTY *prop-name* LEVEL=*level-name* < *options*>;

Details

Required Arguments

prop-name
specifies a valid SAS name for the property. Usually this is the name of a column in the input data source. If it is not the name of a column, then you must include the COLUMN= option to specify the column name. For naming guidelines, see “Naming Guidelines and Rules for the SAS OLAP Server ” on page 51.

LEVEL=*level-name*
specifies the name of the level that you are assigning the property to.

Options

CAPTION=*'string'*
specifies a maximum of 200 characters that can be used to create a meaningful description of the level. Third-party applications that report on cube data might display this description. If the caption includes blank spaces or special characters that are not permitted in a valid SAS name, then enclose the caption within quotation marks. The default value is the column's label if it exists; otherwise tit is he property name.

Note: Cubes that are built with captions that are longer than 200 characters cannot be fully registered in the SAS Metadata Repository. Captions will be truncated to 200 characters.

COLUMN=column-name
specifies the name of a column from the input data source. You must use this option if the column name is not the same as the property name.

DESC | DESCRIPTION=*'string'*
specifies any number of characters that can be used to create a meaningful description of the level. Third-party applications that report on cube data might display this description. If the description includes blank spaces or any characters that are not permitted in a valid SAS name, then enclose the text within quotation marks. The default is the value of the CAPTION= option if one exists; otherwise, the column's label.

ESRI_MAP_FIELD=*MapFieldName*
specifies the field of the map layer to be associated with the level. It must be a map field associated with the LEVEL statement *MapLayerName*. The OLAP property must be named "SAS_SPATIAL_ID" and must be a property of the level associated with the map layer. It must be associated with all hierarchies in the dimension. If the HIERARCHY= (or HIERARCHIES=) option is specified, all hierarchies in the dimension must be specified.

Note: There can be only one map field for each SAS_SPATIAL_ID property. In addition, there can be only one SAS_SPATIAL_ID property per level.

HIERARCHY=(*hier-name ... hier-nameN*)
specifies the name of one or more hierarchies that contain the level. If you do not include the HIERARCHY option, then the property is automatically assigned to all occurrences of the level in all of the hierarchies in which it appears. Otherwise, the property is assigned to the level only in the specified hierarchies.

MLSID=*n*
is a positive integer identifier between 0 and MACINT (2147483647) that identifies the observation in the data set that contains the translated caption and description for the property. This identifier is expected in the MLSID column of the data set specified by DIMTABLECAPPREF=, DIMTABLELIBREF=, and the USER_DEFINED_TRANSLATIONS statement.

Note: For further information on MLS caption tables and MLSID, see "USER_DEFINED_TRANSLATIONS Statement" on page 361.

Associate a Geographical Region with a Level

To associate a geographical region with a level, create a property for that level named SAS_SPATIAL_ID. The spatial property provides field values that are recognized by the Esri mapping service, as shown in the following example:

```
PROPERTY SAS_SPATIAL_ID
LEVEL=Level2_ProvinceID
HIERARCHY= (geography )
COLUMN=Level2_ProvinceID_Char
CAPTION='SAS_SPATIAL_ID'
```

Example

In the following example, the COLUMN= option is used in the first two PROPERTY statements because the column name is different from the property name. In this way, the property named **`Population`** can be assigned to both the **`country`** level and the **`state`** level in the **`geo`** hierarchy. The level **`state`** has two properties: **`Population`** and **`West_of_Miss`**.

```
property Population
   column=p_country
   hierarchy=geo
   level=country
   ;
property Population
   column=p_state
   hierarchy=geo
   level=state
   ;
property West_of_Miss
   hierarchy=geo
   level=state
```

;

MEASURE Statement

Defines the cube's measures and indicates how they map to the input data. Include one MEASURE statement for each measure in the cube. Each cube must have at least one measure. Measure names must be unique. You can have a maximum of 1,024 measures per cube. All cube aggregations have identical measures.

Syntax

MEASURE *measure-name* STAT=*statname* <*options*>;

Details

Required Arguments

measure-name
specifies a valid SAS name for the measure. The name must be unique. For naming guidelines, see “Naming Guidelines and Rules for the SAS OLAP Server ” on page 51.

STAT= *statname*
specifies the statistic for the measure. The following base statistics are available: N, NMISS, NUNIQUE, SUM, MAX, MIN, or USS. In addition, these derived statistics are also available: AVG, RANGE, CSS, VAR, STD, STDERR, CV, T, PRT, LCLM, or UCLM.

Note: At least one non-NUNIQUE measure must be defined.

New cubes that are based on a data source that contains existing summarized data (where such data has been indicated in at least one AGGREGATION statement via the TABLE= option), must include MEASURE statements for the stored statistics required for each derived statistic that you want to create for the new cube. For example, if you want to calculate AVG, you must create measures for N and SUM, as well as AVG.

Note: MOLAP aggregations do not require the N and SUM.

The following table indicates which stored statistics are required for each derived statistic:

Table 12.5 Stored Statistics Required for Each Derived Statistic

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

Note: For information about statistic formulas, see "Keywords and Formulas" in *Base SAS Procedures Guide*.

For cubes that are not loaded from a fully summarized data source (that is, you specified a data source by using the DATA | FACT= option), some statistics use formats taken from the input data source. Specifically, if the statistic is SUM, MIN, MAX, RANGE, AVG, STD, STDERR, LCLM, or UCLM, then PROC OLAP uses the format that is assigned to the column specified by the COLUMN | ANALYSIS= option. The following table lists the formats used for the other supported statistics:

Table 12.6 Default Formats Used for Statistics

Statistic	Format Used
CSS	BEST.
CV	8.2
N	12.0
NMISS	10.0
PRT	6.4
T	7.3
USS	BEST.
VAR	BEST.

For cubes that are loaded from a fully summarized data source (that is, you specified the data source by using the AGGREGATION statement), the default format is BEST12.

To override the default formats, you can either set the FORMAT= option or use a SAS FORMAT statement. The FORMAT= option also overrides a FORMAT

statement. When you rebuild a cube that has been physically deleted, the rebuilt cube still uses the formats originally saved in the cube's metadata. This means that the rebuilt cube does not automatically include any formatting changes that you might have made in the input data source. To manually specify the new formats, edit and rebuild the cube by using SAS OLAP Cube Studio.

COLUMN | ANALYSIS=*variable-name*
specifies the name of a numeric column that is contained in the cube's input data source. (You can use a column as a measure even if it is also being used as a level.)

If the cube is based on an unsummarized data source, then *variable-name* is the name of the column in that data source from which the measure will be calculated. Use COLUMN= to specify the column. If the cube is based on a summarized data source, then *variable-name* can be the name of the numeric column in the data source that was used as the analysis variable for the pre-calculated measure. Use ANALYSIS= to specify the column. It can also be a name that identifies a logical association between measures with the same *variable-name*.

For example, if your cube has three measures, N, SUM, and AVERAGE, and if those measures were derived from the same analysis variable, then you could specify `ANALYSIS=Sales` to logically link the three measures through their shared analysis variable.

As a further illustration, assume that you were building a cube with an NWAY aggregation that was specified using a summarized SAS data set. The data set contains the columns Country, Region, Division, Year, Quarter, Month, SumOfSales, and NumOfSales. You would use two MEASURES statements, one for SumOfSales and another for NumOfSales, as follows:

```
measure Sales_Sum  stat=sum  aggr_column="SumOfSales"  analysis="Sales"
   desc='Sum of Sales'  units='Dollars'  format=dollar10.2 ;
measure Sales_N  stat=n  aggr_column="NumOfSales"  analysis="Sales"
   desc='Number of Sales'  units='Dollars'  format=dollar10.2 ;
```

The Sales column becomes logically linked with the physical columns SumOfSales and NumOfSales.

If the cube consists of a combination of summarized and unsummarized data sources, then *variable-name* refers to both a physical and a logical entity. For example, you might have a cube that requires a physical analysis variable to create a crossing, but that same cube already contains other, higher-level aggregations. In this case, the analysis variable is also used to logically link the measures in the pre-existing aggregations that were derived from the same input column. The default value is the measure name.

An unsummarized data source is specified with the DATA | FACT= option in the PROC OLAP statement. A summarized data source is specified with the TABLE= option in an AGGREGATION statement. The COLUMN argument is not required for the NUNIQUE statistic and will be ignored for the NUNIQUE statistic if specified.

Options

AGGR_COLUMN=*input-column*
specifies the name of the numeric column in the summarized input data that contains the values for the measure. The source of the summarized input data is

specified in the AGGREGATION statement. This option is valid only for stored statistics.

CAPTION=*'string'*
specifies a maximum of 200 characters that can be used to create a meaningful description of the measure. Third-party applications that report on cube data might display this description. If the text includes blank spaces or any characters that are not permitted in a valid SAS name, then enclose the text within quotation marks.

Note: Cubes that are built with captions that are longer than 200 characters cannot be fully registered in the SAS Metadata Repository. Captions will be truncated to 200 characters.

The default is based on the statistic and the COLUMN= value, as shown in the following table. For example, if the statistic is *SUM* and the COLUMN= value is *Sales*, then the default caption is *Sum of Sales*.

Table 12.7 *Defaults for the CAPTION= Option If No Caption Is Specified*

Statistic Used for Measure	Default Caption
AVG	Average *measure-column-name*
CSS	Corrected Sum of Squares of *measure-column-name*
CV	*Measure-column-name* Coefficient of Variation
LCLM	*Measure-column-name* Lower Confidence Limit
MAX	Maximum *measure-column-name*
MIN	Minimum *measure-column-name*
N	Number of Values for *measure-column-name*
NMISS	Number of Missing Values for *measure-column-name*
NUNIQUE	Number of Unique Values for *level-name* in *hierarchy-name*
PRT	Probability of Greater Absolute Value for *measure-column-name*
RANGE	*Measure-column-name* Range
STD	*Measure-column-name* Standard Deviation

Statistic Used for Measure	Default Caption
STDERR	*Measure-column-name* Standard Error of Mean
SUM	Sum of *measure-column-name*
T	*Measure-column-name* T Value
UCLM	*Measure-column-name* Upper Confidence Limit
USS	*Measure-column-name* Uncorrected Sum of Squares
VAR	*Measure-column-name* Variance

DEFAULT
identifies a measure as the default measure for the cube. The default value is the measure defined in the first MEASURE statement

DESC | DESCRIPTION=*'string'*
specifies any number of characters that can be used to create a meaningful description of the measure. Third-party applications that report on cube data might display this description. If the text includes blank spaces or any characters that are not permitted in a valid SAS name, then enclose the text within quotation marks. The default value is caption.

FORMAT=*sas-format-name*
specifies the SAS format to be used to display the value of the measure. This format overrides the default format (see STAT= for more information) and any format that is specified in a SAS FORMAT statement. Both formats supplied by SAS and user-defined formats are supported.

Note: When you rebuild a cube that has been physically deleted, the rebuilt cube still uses the formats that were originally saved in the cube's metadata. This means that the rebuilt cube does not automatically include any formatting changes that you might have made in the input data source. To manually specify the new formats, edit and rebuild the cube by using SAS OLAP Cube Studio.

HIERARCHY=*'string'*
specifies the hierarchy in which the level resides. This option is used only with the NUNIQUE statistic. If there is only one hierarchy, then the option can be omitted.

Note: The HIERARCHY= option will be ignored for non-NUNIQUE statistics if specified.

INCLUDE_CALCULATED_MEMBER | INCLUDE_CALC
specifies that calculated members are included in the NUNIQUE count for the measure statement. This option applies to the NUNIQUE statistic option only.

LEVEL= *'string'*
specifies the level for which a unique count is determined. This option is used only with the NUNIQUE statistic. The default is the measure name.

Note: The LEVEL= option is ignored for non-NUNIQUE statistics if specified.

MLSID=*n*
is a positive integer identifier between 0 and MACINT (2147483647) that identifies the observation in the data set that contains the translated caption and description for the measure. This identifier is expected in the MLSID column of the data set specified by DIMTABLECAPPREF=, DIMTABLELIBREF=, and the USER_DEFINED_TRANSLATIONS statement.

Note: For further information on MLS caption tables and MLSID, see “USER_DEFINED_TRANSLATIONS Statement” on page 361.

NOINCLUDE_CALCULATED_MEMBER | NOINCLUDE_CALC
specifies that calculated members are not included in the MEASURE statement.

UNITS=*'string'*
specifies a maximum of 256 characters that can be used to create a meaningful description of the measure's units (for example, “pounds sterling”). Third-party applications that report on cube data might display this description. If the text includes blank spaces, mixed-case letters, special characters, then enclose the text within quotation marks.

AGGREGATION Statement

Defines an aggregation of the cube based on level information that you provide. You can specify level names that are associated with an unsummarized data source, or you can specify level names that match columns in a table that contains existing aggregated data. The levels can exist in more than one dimension. You do not need to include dimension names, because level names must be unique across dimensions.

Syntax

AGGREGATION *level-name <level-name2 level-name3 ...level-nameN> / <options>*;

Details

Required Arguments

level-name
is the level that is to be used to create the aggregation. Additional level names are optional. Names are separated by spaces. Names are separated from option specifications with a required slash character (/). You do not have to include all levels that are specified in all HIERARCHY statements, but the names that you do specify must match the names that are used in the HIERARCHY statements. You can include a TABLE= option to identify a table that contains existing aggregated information for your specified levels. The levels that you specify must match columns in the input table.

Levels must be listed in drill-path order. You cannot specify an aggregation that contains a summary level that could never be requested. For example, if your TIME hierarchy contains the levels *Year*, *Month*, and *Day*, you could specify *Year* and *Month* as an aggregation, but not *Month* by itself.

Options

Note: For a list of the options that can be used to optimize cube creation and query performance, see "Specialized Options for PROC OLAP" on page 367.

COMPRESS | NOCOMPRESS
specifies whether to store the aggregation table in a compressed format on disk. The default value is NOCOMPRESS.

DATAPATH=(*'pathname' ...'pathnameN'*)
specifies the location of one or more partitions in which to place aggregation table data. The data is distributed by cycling through each partition location according to the partition size. This is set by using the PARTSIZE= option. For example, if you specify `DATAPATH=('C:\data1' 'D:\data2')`, then PROC OLAP places the first partition of the aggregation table into directory `C:\data1`, the second partition of the table into directory `D:\data2`, the third partition of the table into `C:\data1`, and so on. It is also possible to have aggregation tables that use fewer than the specified number of partitions. For example, your aggregation table might fit entirely into `C:\data1`.

The default value is the cube subdirectory of the location that is specified by the PATH= option in the PROC OLAP statement.

INDEX | NOINDEX
specifies whether to create the specified aggregation with indexes. For faster cube creation and updates, you can set this option to NOINDEX. However, the lack of indexes might adversely affect query performance. The default value is INDEX.

Note: Indexes are not created for aggregations that have fewer than 1,024 records.

INDEXPATH=(*'pathname' ...'pathnameN'*)
specifies the locations of the index component files that correspond to each aggregation table partition as specified by the DATAPATH= option.

NAME=*'aggregation-name'*
specifies a maximum of 256 characters as the name of the aggregation. If the name includes blank spaces or any characters that are not permitted in a valid SAS name, then the name must be enclosed within quotation marks. The name is stored with the cube's metadata. The default value is a name assigned by SAS, such as AGGR1.

PARTSIZE=*partition-size*
specifies the partition size in megabytes of the aggregation table partitions and their corresponding index components. The default value is a

SEGSIZE=*rows-per-segment*
specifies the number of observations (table rows) in the file segment of the index component. The value is expressed in multiples of 1,024. The minimum value is 1 (1,024 rows). The segmented indexes are used to optimize the processing of WHERE expressions. Each parallel thread is given a segment of the table to evaluate that is equal to the value of the SEGSIZE= option multiplied by 1,024. The default value is 8 (8 x 1,024 = 8,192 rows).

The value of this option overrides for the current aggregation any such value that was specified for all aggregations in the PROC OLAP statement.

TABLE=*libname.dataset*
specifies the name of a SAS data set or data view that contains the data for one aggregation. Every level that is listed in the AGGREGATION statement must match a column that contains aggregation information in the specified table. Place this option after the list of level names.

Analysis columns in the table are mapped to the numeric columns that are specified with the AGGR_COLUMN= option in MEASURE statements.

You can also set data set options with the TABLE= option. Options are stored within the cube and reapplied when the data is accessed at run time. For more information, see “Data Set Options” in *SAS Language Reference: Concepts*. You cannot use the TABLE= option in an AGGREGATION statement that is used to add an aggregation to an existing cube.

Example

Here is an example of an AGGREGATION statement that specifies three levels and uses the NAME= option. The slash character (/) is required to separate level names from option specifications.

```
aggregation country prodtype year /
   name='Product Types by Country';
```

DROP_AGGREGATION Statement

Removes an aggregation from the specified cube. You can specify the levels that are in the aggregation, or the name of the aggregation, or both the levels and the name. The slash character (/) is required to separate level names from option specifications.

Syntax

DROP_AGGREGATION *level-name1 < level-name2 ...level-nameN>* / NAME=*aggregation-name*;

Details

Required Arguments

At least one of the following arguments is required for a DROP_AGGREGATION statement:

level-name1
specifies the names of the level that is in the aggregation that you want to drop. Additional levels can be specified using blank spaces to separate the level names, as shown in the following example. `drop_aggregation Year Month Product / name=Sales ;`

NAME=*'aggregation-name'*
specifies the name of the aggregation that you want to drop. If the name includes blank spaces or any characters that are not permitted in a valid SAS name, then the name must be enclosed within quotation marks.

DEFINE Statement

Defines a global calculated member or a named set for any cube that is registered in a SAS Metadata Repository.

Syntax

DEFINE MEMBER | SET *'member-or-set-name'* AS *'mdx-expression'* ;

Details

Overview

The DEFINE statement is used to create a global calculated member or a named set. Only the registration of the member is stored; the value is calculated when a query is submitted. A named set is an alias for a specified MDX expression that returns a member or set. Named sets are often used to make complex MDX queries easier to read and maintain.

The defined calculated members and named sets are available to any session that creates a query in the context of the SAS OLAP Server and the schema defined in the METASRV statement of the PROC OLAP code that is used to create the global member or set.

DEFINE statements can apply to more than one cube, so the CUBE= option is not required to use this statement. The METASVR statement verifies that the cube registration exists in the metadata repository.

Required Arguments

MEMBER | SET
indicates whether you are creating a calculated member or a named set.

'member-or-set-name'
specifies the name of the member or set that you are creating. If you are creating a calculated member, then this value specifies a name for the member that will be calculated by the MDX expression. If you are creating a named set, then this value is the alias for the specified MDX expression.

AS *'mdx-expression'*
specifies the MDX expression.

Options

MLSID=*n*
is a positive integer identifier between 0 and MACINT (2147483647) that identifies the observation in the data set that contains the translated caption and description for the calculated member or named set. This identifier is expected in the MLSID column of the data set specified by DIMTABLECAPPREF=, DIMTABLELIBREF=, and the USER_DEFINED_TRANSLATIONS statement.

Note: For further information on MLS caption tables and MLSID, see "USER_DEFINED_TRANSLATIONS Statement" on page 361.

Note: The MLSID=n option is part of the syntax of the MDX Expression, as shown in the following example.

```
define Member "[&testname].[date].[all date].[June]" as '6, mlsid=11'
```

VISUALTOTALS_BEHAVIOR | VISUALTOTALS_BEHAVIOR= | |
The VISUALTOTALS_BEHAVIOR option enables you to clear total rows, subtotal rows, or both total and subtotal rows. This option can be added to MDX calculated measure definitions wherever they are defined. You can define MDX calculated measures with the PROC OLAP code DEFINE statement and the SAS OLAP Cube Studio Calculated Members wizard.

Here is the syntax for this option:

```
VisualTotals_Behavior|VisualTotal_Behavior=(BLANK | BLANK_SUBTOTALS |
BLANK_TOTALS, {dimension_name|hierarchy_name}* )
```

| |

For further information, see "OLAP VISUALTOTALS_BEHAVIOR Option" on page 103 .

Example

The DEFINE statement can be used alone as shown in the following example, which defines two calculated members and one named set. The METASVR is the only other required statement. To define multiple sets or calculated members, separate option values with a comma.

```
proc olap;
metasvr olap_schema='Services Schema'
   repository='services'
   host='misdept.us.mar.com'
   port=9999
   userid=jjones
   pw='my password'
   ;
 define member '[mddbcars].[Measures].[avg]' as
             '[Measures].[sales_sum]/[Measures].[sales_n]',
      member '[sales].[Measures].[stat1]' as
             '[Measures].[qty] +1',
      set '[campaign].[myset]' as
          '[campaign_dates].[All campaign_dates].children'
   ;
run;
```

The DEFINE statement can also be used with a PROC OLAP program that creates a cube or with a program that adds aggregations to or deletes aggregations from an existing cube. Cube builds, additions, and deletions occur before the DEFINE statement is processed, so the DEFINE statement is not processed if those statements fail.

```
proc olap data=olapsio.cars
   cube=mddbcars
   path='d:\services\'
   ;
metasvr olap_schema='Services Schema'
   repository='cars'
   host='misdept.us.mar.com'
   port=9999
   userid=jjones
   pw='my password'
   ;
dimension date
   hierarchies=(date)
   sort_order=ascending
   ;
hierarchy date
   LEVELS=(dte)
   ;
level 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 sales_sum
   column=sales
   stat=sum
   format=dollar15.2
   ;
measure sales_n
   column=sales
   stat=n
   format=12.0
   ;
define member '[mddbcars].[Measures].[avg]' as
   '[Measures].[sales_sum] / [Measures].[sales_n]'
   ;
run;
```

UNDEFINE Statement

Deletes from a SAS metadata repository one or more global calculated members or named sets. To delete multiple calculated members or named sets in a single UNDEFINE statement, use commas to separate instances of MEMBER | SET 'member-or-set-name'.

Syntax

UNDEFINE MEMBER | SET *'member-or-set-name'* ;

Details

Required Arguments

MEMBER | SET
indicates whether you are deleting a global calculated member or a named set.

'member-or-set-name'
specifies the name of the member or set that is to be deleted from the metadata repository. Cube names and dimension names are required for each *member-or-set-name*. Square brackets ([ ]) are optional inside the quotation marks of *member-or-set-name*, as shown in the preceding example.

Example

The following example shows how a single UNDEFINE statement can be used to delete from a metadata repository two calculated members and one named set.

```
proc olapl;
metasvr olpa_schema='Services Schema'
   repository='services'
   host='misdept.us.mar.com'
   port=9999
   userid=jjones
   pw='my password'
   ;
undefine member '[carsCube].[Measures].[avg]',
   member '[sales].[Measures].[stat1]',
   set '[campaign].[myset]'
   ;
run;
```

USER_DEFINED_TRANSLATIONS Statement

Specifies the locales that are associated with the data sets that you specify in the DIMENSION statement. The USER_DEFINED_TRANSLATIONS statement is required to use the Multiple Language Support capabilities of the SAS OLAP Server. Alternative statement names are UDT and USER_DEFINED_TRANSLATION.

Syntax

USER_DEFINED_TRANSLATIONS | **UDT** *locale* <*locale2 ...localeN*> ;

Details

Overview

PROC OLAP uses the UDT statement information, along with DIMENSION statement options, to read your alternate locale data sets and create locale-specific metadata for use at query time. Query results are returned in the language of the requested locale. The Multiple Language Support feature is available only for cubes that are loaded from a star schema. The alternate locale data set names consist of a prefix, which indicates the member, and a suffix, which indicates the language. The UDT statement supplies the suffix. The DIMTABLEMEMPREF= option in the DIMENSION statement specifies the member prefix. For example, if the member prefix is **`dealdim_`** and the suffix is **`pl_PL`**, then PROC OLAP looks for a data set named **`dealdim_pl_PL.sas7bdat`** in the library that is specified by the DIMTABLELIBREF= option.

The default locale is the first locale specified in the UDT statement. Data sets in the default locale do not use the suffix that is defined by the UDT statement.

Required Argument

locale

specifies the locales that correspond to the data sets contained in the library that is specified by the DIMTABLELIBREF= option in the DIMENSION statement. Separate locales with a space. Locales are specified using the standard Posix five-character notation, as specified in "LOCALE= Values and Default Settings for ENCODING, PAPERSIZE, DFLANG, and DATESTYLE Options," in the *SAS 9.4 National Language Support (NLS): Reference Guide*. Your cube can specify any of the locales that are valid for the SAS system option LOCALE=.

Note The first locale specified is the default locale of your cube. This default locale must match the locale of your SAS session.

SAS Servers and Character Encoding

If your server metadata contains characters other than those typically found in English, then you must be careful to start your server with an ENCODING= or LOCALE= system option that accommodates those characters. For example, a SAS server started with the default US English locale cannot read metadata that contains Japanese characters. SAS will fail to start and will log a message indicating a transcoding failure.

In general, different SAS jobs or servers can run different encodings (such as ASCII/EBCDIC or various Asian DBCS encodings) as long as the encoding that is used by the particular job or server can represent all the characters of the data being processed. In the context of server start up, this requires that you review the characters used in the metadata describing your server (as indicated by the SERVER= object server parameter) to ensure that SAS runs under an encoding that supports those characters.

Example

The following sample code looks for these dimension data sets in the **mylib** library. In this example, Polish is the default locale, so the suffix is not used.

```
dimension date
   hierarchies=(date)
   sort_order=ascending
   dimtablelibref=mylib
   dimtablempref=ctimedim_
   factkey=dte
   dimkey=dte
   ;
hierarchy date levels=(dte)
   ;
level dte
   ;
dimension cars
```

```
   hierarchies=(cars)
   sort_order=ascending
   dimtablelibref=mylib
   dimtablempref=cardim_
   factkey=carkey
   dimkey=carkey
   ;
dimension cars
   levels=(car color)
   ;
dimension dealers
   hierarchies=(dealers)
   sort_order=ascendng
   dimtablelibref=mylib
   dimtablempref=dealdim_
   factkey=dealerkey
   dimkey=dealerkey
   ;
hierarchy dealers
   levels=(dealer dest)
   ;
user_defined_translations
   pl_PL   /* Polish as used in Poland; default locale */
   en_US   /* English as used in the United States */
   ja_JP   /* Japanese as used in Japan */
   ;
```

The following table shows the expected data set names for each of the locales specified in the UDT statement.

Table 12.8 *Locales and Associated Data Set Names*

Locale	Default	Dimension Data Sets		
English	No	ctimedim_en_US	cardim_en_US	dealdim_en_US
Japanese	No	ctimedim_ja_JP	cardim_ja_JP	dealdim_ja_JP
Polish	Yes	ctimedim_	cardim_	dealdim_

REORGANIZE_LEVEL Statement

Enables you to reorganize individual eligible levels for a cube. It is used when you have performed multiple cube updates on a cube such that the levels values must be reorganized. Any level used in this statement must have been updated with new captions during prior instances of cube update.

Syntax

REORGANIZE_LEVEL | REORG_LEVEL *level-name*

Example

Here is an example of the REORGANIZE_LEVEL statement:

```
PROC OLAP;
DIMENSION CUSTOMER SHARED;
REORG_LEVEL CUSTOMER_NAME;
RUN;
```

Usage: OLAP Procedure

Maintaining Cubes

Building a Cube from an Existing Registration

It is possible to have cube argumentDescriptions in the SAS metadata that do not contain the associated physical cubes. For example, you can use the DELETE_PHYSICAL= option in the PROC OLAP statement to delete a cube but leave its argumentDescription intact. You can also use SAS OLAP Cube Studio to save only the argumentDescription of a new cube.

The following table lists the PROC OLAP statements and options that you use to build a cube from an existing metadata argumentDescription:

Table 12.9 *Statements and Options Used to Build a Cube from an Existing Registration*

Statements	Options
PROC OLAP	CUBE=
METASVR	OLAP_SCHEMA=

Adding Aggregations to an Existing Cube

The following table lists the PROC OLAP statements and options that you use to add aggregations to an existing cube.

Table 12.10 *Statements and Options Used to Add Aggregations to an Existing Cube*

Statements	Options	Required or Optional?
PROC OLAP	CUBE=	Required
METASVR	OLAP_SCHEMA=	Required
AGGREGATION	Names of the contiguous levels to be used to create the aggregation	Required
	NAME=	Optional
	DATAPATH=	Optional
	INDEXPATH=	Optional
	COMPRESS | NOCOMPRESS	Optional
	INDEX | NOINDEX	Optional
	PARTSIZE=	Optional
	SEGSIZE=	Optional

Note: You can add and delete aggregations in the same PROC OLAP script.

Note: You cannot add aggregations to a cube that contains aggregated data from a fully summarized data source.

Deleting Aggregations from an Existing Cube

The following table lists the PROC OLAP statements and options that you use to delete aggregations from an existing cube.

Table 12.11 *Statements and Options Used to Drop Aggregations from an Existing Cube*

Statements	Options
DROP_AGGREGATION	Specify one or more level names that correspond to the aggregations that you want to remove, or use the aggregation name to specify the aggregation that you want to remove.
METASVR	OLAP_SCHEMA=
PROC OLAP	CUBE=

Note: You can add and delete aggregations in the same PROC OLAP script.

Note: You cannot delete aggregations from a cube that contains aggregated data from a source other than the input data source.

Deleting Cubes

The following table lists the PROC OLAP statements and options that you use to delete existing cubes.

If you use the DELETE option, then both the physical cube and its argumentDescription, which is stored in the metadata server, are deleted.

Table 12.12 *Statements and Options Used to Delete a Cube and Its Metadata*

Statements	Options
METASVR	OLAP SCHEMA=
PROC OLAP	CUBE=
	DELETE

If you use the DELETE_PHYSICAL option, then only the physical cube is deleted; the argumentDescription remains intact.

Table 12.13 Statements and Options Used to Delete a Cube but Retain Its Metadata

Statements	Options
METASVR	OLAP_SCHEMA=
PROC OLAP	CUBE=
	DELETE_PHYSICAL

Specialized Options for PROC OLAP

Options for Managing Ragged Hierarchies

If a hierarchy is balanced, then all of its branches descend to the same level, and each member has a parent level that is positioned immediately above it. However, hierarchies are not always balanced, and sometimes they contain missing hierarchy members. To manage missing hierarchy members, you can use these four options, which were created specifically for ragged hierarchies:

Table 12.14 Options That Can Be Set to Manage Missing Hierarchy Members in Ragged Hierarchies

Options	Statements Where Option is Available
EMPTY_CHAR=	PROC OLAP and HIERARCHY
EMPTY_NUM=	PROC OLAP and HIERARCHY
EMPTY=	LEVEL
IGNORE_EMPTY	HIERARCHY and LEVEL

If EMPTY= or IGNORE_EMPTY are specified, then the values of EMPTY_CHAR= and EMPTY_NUM= are ignored within the scope of the hierarchy or level.

Options Used for Performance

When you create a cube, you can set some options that can be used to optimize cube creation and query performance. If you set the options in the PROC OLAP statement, then the settings are applied to all aggregations in the cube. If you set the options in the AGGREGATION statement, then the options apply to that specific

aggregation. Options set for individual aggregations override any options set in the PROC OLAP statement.

Here are the options:

- ASYNCINDEXLIMIT=
- COMPACT_NWAY
- COMPRESS | NOCOMPRESS
- CONCURRENT=
- DATAPATH=
- INDEXPATH=
- INDEXSORTSIZE=
- MAXTHREADS=
- INDEX |NOINDEX
- PARTSIZE=
- SEGSIZE=

Note: ASYNCINDEXLIMIT=, CONCURRENT=, INDEXSORTSIZE=, and MAXTHREADS= are available only in the PROC OLAP statement.

For an explanation of these options, see the PROC OLAP statement and the AGGREGATION statement.

Understanding Shared Dimensions

Overview

Shared dimensions are dimensions that are used by multiple cubes. They are advantageous because they enable you to define the dimension once, rather than redefine the dimension for every cube that uses it. A shared dimension can be updated once, and all affected cubes can be made aware of the change. This saves both processing time to build the dimension and disk space that is used to store the member information.

Here are some of the features of using a shared dimension:

- Shared dimensions can be defined once and then referenced by multiple cubes.
- Identifying which cubes reference a given dimension is easy.
- A shared dimension can be updated in one place, and all cubes that reference it are made aware of the change.
- Cubes can include both shared and private dimensions.
- Cubes referencing shared dimensions can remain online as new members are added to the dimension.

- You can query the shared dimension for its member structure without having it be part of a cube.

Using shared dimensions enables an enterprise to have a common definition for a dimension that all cubes in the enterprise can use. This provides consistency across the enterprise with a level of transparency that maximizes efficiency. If you are building a SAS 9.3 or later ROLAP cube, you can implement shared dimensions and incrementally update the cube with new information from the cube's dimension tables. This provides an advantage in that you don't have to rebuild the cube from scratch simply to add new members to a dimension.

Shared dimensions do not support multiple languages.

Functionality

A shared dimension is defined in PROC OLAP or in SAS OLAP Cube Studio. The definition includes all hierarchies, all levels, and all member properties known to the dimension. The data source for a shared dimension is a single dimension table (or a view of multiple physical tables).

Use of shared dimensions is governed by the following functionality rules:

- A shared dimension can be referenced only in a star schema or in a hybrid star schema load.
- Shared dimensions are grouped together in an OLAP schema, just as cubes are. However, a cube can reference a shared dimension in a different OLAP schema.
- When a shared dimension is defined, all hierarchy member trees, caption files, and property files are built. Cubes that use the shared dimension reference these files directly, both to build the cube and during query processing. It is also possible to define the metadata and build the physical files later.

Effective use of shared dimensions relies on the ability to take advantage of OLAP features such as dimension updates, dimension deletions, multiple language services, and security and authorization rules. The following table describes each of these features with respect to shared dimensions.

OLAP Feature	Shared Dimension Description
Dimension Update	Updates to a shared dimension include the following: - new members within the existing structure - changes to the existing structure - additional hierarchies and levels - reorganization of member ordinals The level of change required in existing cubes referencing the dimension depends on what type of update is made. For example, new members in the dimension without new data in a fact table do not require updates to the cube aggregations. Changes to the existing structure and additional hierarchies and levels not included in the cube's definition require that the entire cube be rebuilt, as is currently the case for such updates in SAS OLAP Cube Studio. Level reorganization requires that the cube aggregations be rebuilt.

OLAP Feature	Shared Dimension Description
Dimension Deletion	When a cube is deleted, the files associated with any shared dimensions are not deleted. The metadata definition for the dimension is updated to indicate that the cube is no longer using the dimension. PROC OLAP syntax and windows in SAS OLAP Cube Studio enable dimension deletion. If existing cubes are still referencing the shared dimension, the dimension deletion is not permitted. The referencing cubes must be deleted first (the procedure displays the names of all affected cubes). To remove the physical files for a shared dimension (that is, DELETE_PHYSICAL), the physical files for all cubes referencing that shared dimension must also be removed. It is not necessary that the cubes be completely deleted, but a DELETE_PHYSICAL must be run against those cubes before it can be run against the shared dimension.
Security and Authorization	Because there is only one metadata registration for shared dimensions and because permissions are added to the metadata, then permissions applied to a shared dimension apply to all cubes using that dimension. Permissions on a shared dimension are inherited from its folder. Changes to the permissions are applied to all cubes using that dimension as well. This reduces the burden on the administrator when applying security for a site. For example, if a particular set of users is not allowed to see data for the members in a particular geographic area such as Canada, it would make sense that those users should see a consistent view (no Canadian data) across all cubes to which they have access. The administrator would still be able to deny access to an entire cube, which would override any permissions set on a shared dimension for that cube. Authorizations on a shared dimension are set in either SAS Management Console or SAS OLAP Cube Studio.

Developing and Managing Shared Dimensions

Creating a Shared Dimension

As with cubes, there are three ways to build a shared dimension with PROC OLAP:

- Long form: This option builds the metadata registration and physical files from statements specifying hierarchies, levels, and so on.

- Short form: This option builds the physical files from an existing metadata registration. SAS OLAP Cube Studio uses this syntax to build shared dimensions.
- REGISTER_ONLY: This option on the procedure statement builds only the metadata registration.

The following example shows the syntax for the METASVR statement using the REGISTER_ONLY option.

```
PROC OLAP REGISTER_ONLY;
  METASVR OLAP_SCHEMA=schema-name <options>;
```

OLAP_SCHEMA=*schema-name*
specifies the OLAP schema that will contain the shared dimension.

options
specify the metadata server connection options.

The following example shows the syntax for specifying a shared dimension and the location of its metadata.

```
DIMENSION [dim-name | "/folder/dim-name"] SHARED PATH="path-name" <options>
```

dim-name | "*/folder/dim-name*"
specifies the name of the shared dimension. *dim-name* is the dimension name, and *"/folder/.."* specifies the fully qualified metadata folder path in which the dimension's metadata should be stored. If the folder path is omitted, the shared dimension metadata is stored in the OLAP schema's metadata folder.

SHARED
indicates that this is a shared dimension.

PATH="*path-name*"
specifies the physical or logical path to the location of a new dimension. Within the specified path, the dimension is stored in a directory that uses the name of the dimension. For example, if you enter the path 'C:\shared_dimensions' and the dimension name is Customer, then the dimension is stored in 'C:\shared_dimensions\Customer'. (If the folder already exists, PROC OLAP will generate a unique pathname based on that path.) Enclose the path within quotation marks.

options
specify the metadata server connection options. All other dimension options are supported for shared dimensions, with the exception of FACTKEY=. This option is used when a shared dimension is included in a cube definition. (See USE_DIMENSION statement.) Use of the FACTKEY= option is ignored.

The following statements complete the definition of the shared dimension and are otherwise the same as those used with private dimensions.

```
HIERARCHY hier-name LEVELS=(lev-name...lev-nameN)<options>;

LEVEL lev-name <options>;

PROPERTY prop-name LEVEL=lev-name <options>;
```

Updating a Shared Dimension

The following is the syntax for updating a shared dimension.

```
DIMENSION dim-name SHARED UPDATE_DIMENSION= [ MEMBERS | MEMBERS_AND_PROPERTIES ]
        [DIMTABLE=dim-table];
```

UPDATE_DIMENSION= [MEMBERS | MEMBERS_AND_PROPERTIES]
causes the specified shared dimension to be rebuilt based on the current contents of the dimension table and the current contents of the OMR metadata. If the MEMBERS_AND_PROPERTIES keyword is specified, then existing members will have their properties updated as well.

[DIMTABLE=*dim-table*]
allows a temporary input data source to be used for the update. This does NOT change the dimension table defined with the shared dimension's metadata registration.

Any cubes using the shared dimension can be disabled during the update by the server administrator. When the cube is enabled, the new members and properties are available to the server. If the cube is not disabled, then the new members are visible as new sessions, or new queries cause the cube to be reopened. (This is similar to the in-place cube update facility.) No data is associated with the new members until the cube itself is updated with new facts. NUNIQUE measure values change with the addition of the new members.

Changing Captions and Descriptions

The following syntax is used to change captions or descriptions.

```
PROC OLAP UPDATE_DISPLAY_NAMES

METASVR <options>;

DIMENSION dim-name SHARED [CAPTION=<caption>][DESCRIPTION=<desc>];

HIERARCHY hier-name [CAPTION=<caption>][DESCRIPTION=<desc>];

LEVEL level-name [CAPTION=<caption>][DESCRIPTION=<desc>];

PROPERTY prop-name [CAPTION=<caption>][DESCRIPTION=<desc>];
```

UPDATE_DISPLAY_NAMES
indicates that the shared dimension's captions and descriptions should be updated. Unlike updating a cube, updating a shared dimension's captions and descriptions does not require the shared dimension to be locked. Thus cubes that use the shared dimension can be queried while its captions and descriptions are updated. As with new members, these updates become visible to cubes as new sessions or queries cause the cube to be reopened.

Reorganizing a Shared Dimension

The following is the syntax for reorganizing a shared dimension.

```
PROC OLAP REORGANIZE_LEVELS;
METASVR <options>;
DIMENSION [ dim-name | "/folderpath/dim-name" ] SHARED;
```

REORGANIZE_LEVELS
causes the captions for all levels contained in the shared dimension identified by the DIMENSION statement to be reorganized. The member trees for the hierarchies are rebuilt. This is an online feature. That is, cubes can continue to be online during the reorganization.

REORGANIZE_LEVEL *level-name* ;
reorganizes only the specified level. This statement can be used as an alternative to the REORGANIZE_LEVELS option.

When the reorganization is complete, a list of all cubes that use the dimension and need to be synchronized is displayed.

All cubes using a shared dimension whose levels have been reorganized continue to reference the previous version of the shared dimension until their aggregations have been synchronized.

If a second reorganization is attempted on the shared dimension before a cube has been synchronized to the first one, then PROC OLAP shows an error and displays a list of the cubes affected. No other updates to the shared dimension are allowed as long as there are any cubes that still need to be synchronized. No updates to a cube are allowed until its aggregations have been synchronized. The cube can instead be rebuilt, using either the long form or short form, at which time it will again have access to all members of the shared dimension. Or it can be deleted.

Deleting a Shared Dimension

The following is the syntax for deleting a shared dimension.

```
PROC OLAP [ DELETE | DELETE_PHYSICAL ];
METASVR <options>;
DIMENSION dim-name SHARED;
```

DELETE
deletes the specified shared dimension.

DELETE_PHYSICAL
deletes only the physical files associated with the dimension and leaves the metadata registration. In either case, however, if there are cubes that are referencing this dimension, the deletion is disallowed and the names of the cubes using the dimension are displayed.

Using a Shared Dimension in a Cube

The following is the syntax for using a shared dimension in a cube.

```
PROC OLAP CUBE=cube-name (FACT|DATA)=fact-table <options>;
METASVR <options>;
USE_DIMENSION [dim-name | "/folder/dim-name"] [OLAP_SCHEMA=schema-name]
        FACTKEY=fact-table column;
```

USE_DIMENSION [*dim-name* | "*/folder/dim-name*"]
specifies a shared dimension to be included in this cube.

OLAP_SCHEMA=*schema-name*
: specifies the OLAP schema that will contain the shared dimension. The schema option is required if shared dimension's schema is different from the schema specified in the METASVR statement.

FACTKEY= *fact-table column*
: provides the name of the key fact table column that corresponds to the DIMKEY key column in the dimension table. This option is required.

HIERARCHY, LEVEL and PROPERTY statements are not needed (or allowed) for those objects in the dimension. They have already been defined and are implied by the use of the dimension in the cube.

If the EMPTY_CHAR or EMPTY_NUM option is specified in the PROC OLAP statement, it does not apply to any shared dimensions used in the cube. It only affects private dimensions defined for the cube.

Synchronizing a Cube After Reorganizing a Shared Dimension

The following is the syntax for synchronizing a cube after reorganizing a shared dimension.

```
PROC OLAP CUBE=cube-name [SYNC_AGGRS | SYNCHRONIZE_AGGRS] <options>;
METASVR <options>;
```

SYNC_AGGRS
: indicates that the cube aggregations will be checked for consistency with the current members trees for all shared dimensions that it uses. If the cube requires synchronization, the cube aggregations are rebuilt.

Chapter 13

OLAPOPERATE Procedure

Overview: OLAPOPERATE Procedure

What Does the OLAPOPERATE Procedure Do?

The OLAPOPERATE procedure enables users to manage an OLAP server, its cubes, its sessions, and their queries. It is intended as a batch interface for many of the functions provided in the OLAP Server Monitor plug-in for SAS Management Console. More specifically, you can use PROC OLAPOPERATE to connect to an OLAP server and perform the following tasks:

- list sessions, queries, and cubes for an active OLAP server
- close individual active sessions
- cancel or close individual open result sets
- disable a cube or all cubes
- enable a cube or all cubes
- refresh a cube or all cubes
- stop, quiesce, pause, or resume the OLAP Server/OLAP Server Cluster

You can establish a connection to a running SAS OLAP Server by providing connection information about the PROC OLAPOPERATE statement or by running the CONNECT statement. All other OLAPOPERATE statements require a connection to an OLAP server. After you have connected, all subsequent statements apply to that server until a DISCONNECT or STOP SERVER statement is executed.

If a connection is made to a secured OLAP server, you must have administrative privileges defined with your login information in the metadata in order to execute these tasks. The OLAPOPERATE procedure is an interactive procedure and its statements are executed as they are encountered. The OLAPOPERATE procedure is terminated by a QUIT or a RUN statement. For example, you can list sessions for a server and then close certain sessions based on the output of the LIST SESSIONS statement, without having to reinvoke the procedure (and reconnect to the OLAP server).

Syntax: OLAPOPERATE Procedure

PROC OLAPOPERATE [<*connection options*>];
CONNECT <*connection options*>;
DISCONNECT;

LIST SESSIONS [CUBE=*cube-name*] [USER=*user-id*] [OUT=*data-set*] [NODEONLY];

LIST QUERIES [SESSION=*owner*] [OUT=*data-set*] [NODEONLY];

LIST ROWSETS [SESSION=*owner*] [OUT=*data-set*] [NODEONLY];

LIST CUBES [CUBE=*cube-name*] [OUT=*data-set*];

CLOSE SESSION *id* | _ALL_ [USER=*user-name*] [CUBE=*cube-name*] [INACTIVE=*seconds*];

CANCEL QUERY *id* | _ALL_ [INACTIVE=*seconds*];

DISABLE CUBE *cube-name* | _ALL_;

ENABLE CUBE *cube-name* | _ALL_;

REFRESH CUBE *cube-name* | _ALL_;

STOP SERVER;

QUIESCE SERVER;

PAUSE SERVER;

RESUME SERVER;

STOP CLUSTER;

QUIESCE CLUSTER;

PAUSE CLUSTER;

RESUME CLUSTER;

Usage: OLAPOPERATE Procedure

OLAPOPERATE and SAS Releases

The OLAPOPERATE procedure can be used on SAS 9.1 through SAS 9.4 OLAP servers. For OLAP servers earlier than SAS 9.4, some of the statements and options are not available.

The following statements and options are available in SAS 9.3 and later releases.

- LIST CUBES
- the OUT= option that applies to LIST SESSIONS, LIST QUERIES, and LIST ROWSETS statements
- the CUBE=, INACTIVE=, and USER= options that apply to the CLOSE SESSION statement
- the INACTIVE= option that applies to the CANCEL QUERY statement

These statements and options are available in SAS 9.4.

- PAUSE CLUSTER
- QUIESCE CLUSTER
- RESUME CLUSTER

- STOP CLUSTER
- the NODEONLY option that applies to LIST SESSSIONS, LIST QUERIES, and LIST ROWSETS statements

Connecting to an OLAP Server

The CONNECT command enables you to establish a connection to a running SAS OLAP Server. You can have only one connection to an OLAP Server at a time.

CONNECT <*connection options*>;

The following server connection options establish communication with a SAS OLAP Server.

HOST=
specifies the host name or network IP address of the computer hosting the SAS OLAP Server (for example, HOST=d1234.na.sas.com). The value `localhost` can be used if the SAS session is connecting to a server on the same computer.

LOGICAL= |
specifies the logical name of the SAS OLAP server that is accessed with the CONNECT command. Beginning with SAS 9.4M1, this option enables you to connect to an OLAP server with the logical server name, rather than the host name and port number. The HOST and PORT options can be used to establish a connection to a running SAS OLAP Server on a designated host. However, if the specified host is down, the connection cannot be established. For load-balanced environments that have multiple hosts or ports, specifying the hosts with the LOGICAL= option enables you to continue the connection process with the next available host in the grid and access the SAS OLAP server.

PASSWORD= |
specifies the password that corresponds to the user ID.

PORT=
specifies the TCP port to which the SAS OLAP Server listens for connections (for example PORT= 5451).

Note: COM connection support was discontinued in SAS 9.3.

PROTOCOL=
specifies the network protocol for communicating with the SAS OLAP Server. The default value is BRIDGE.

Note: COM connection support was discontinued in SAS 9.3.

SSPI
(Security Support Provider Interface) specifies that the identity of the user running the SAS session will be used to establish the connection. The USERID and PASSWORD options are unnecessary and are ignored when SSPI is specified.

Restriction This connection option is used only when connecting to an OLAP server running on a Windows machine.

SPN=
specifies the Service Principal Name of the connection destination. This option is optional even if SSPI is specified. When SPN is not specified, one is created using the HOST= and PORT= values.

Restriction This connection option is used only when connecting to an OLAP Server running on a Windows machine.

USERID=
specifies a user ID to be used to connect to the SAS OLAP server. The user must have Administer privileges for the OLAP Server in order to execute statements of the OLAPOPERATE procedure.

Disconnecting from an OLAP Server

You can disconnect from the current SAS OLAP server with the DISCONNECT statement. After you have disconnected from the server, the only other statement that you can execute is the CONNECT statement to connect to another (or the same) server.

When the OLAPOPERATE procedure is terminated with the QUIT or the RUN statement, an automatic server disconnection is executed. It is not necessary to explicitly disconnect from the server before terminating the procedure.

DISCONNECT;

Listing Active Sessions on an OLAP Server

You can list information about active sessions on the OLAP Server with the LIST SESSIONS statement.

LIST SESSIONS [CUBE=*cube-name*] [USER=*user-id*] [OUT=*data-set*] [NODEONLY];

If the CUBE option is specified, the output contains information about all sessions currently accessing the specified cube. If the USER option is specified, the output contains information about all sessions for the specified user. If neither option is specified, then all active sessions are displayed.

If the OUT option is specified, the output is written to the specified data set.

The LIST SESSIONS statement generates the following information:

Session Description
is the description assigned to the session.

Session ID
is the unique session identifier.

Session Name
is the name assigned to the session.

Session Owner
is the user ID of the session owner.

Seconds Inactive
is the number of seconds since the last query for this session.

Results Sets Open
is the number of open results sets for the session.

The following information is available for clients connecting to a SAS 9.2 or later version of the OLAP server:

Cube Names
presents the names of the cubes being used by this session.

Here is an output example for LIST SESSIONS.

```
Session ID: C3144659-F1B5-48E1-84D0-48DC439946A3
Session owner: olaptst1@CARYNT
Session name:
Session description:
Session seconds inactive = 69 and open results = 3
```

For clients connected to a SAS 9.2 or later OLAP server, the cubes that are active for the session are also listed.

```
Cube CENSUSCMR is active in this session.
Cube MDDBCARS is active in this session.
```

Listing Session Queries on an OLAP Server

You can list information about session queries with the LIST QUERIES statement.

LIST QUERIES [SESSION=*owner*] [OUT=*data-set*] [NODEONLY];

If the SESSION option is specified, then the queries associated with a particular user are listed. Otherwise, all queries are listed. If the OUT option is specified, the output is written to the specified data set.

The LIST QUERIES statement generates the following information:

MDX String
contains the MDX string for the query. The cube referenced in the query is easily determined because the FROM clause is placed at the front of the query. For example, an MDX query such as `select measures.members on columns from mddbcars` is displayed as `from mddbcars select measures.members on columns`.

Results Set Type
indicates the type of the query's results set. Valid values for this item are Multidimensional or Flattened Rowset.

Size
indicates the approximate memory used in the query.

Note: For SAS versions earlier than 9.2, this item indicates the number of cells returned in the query.

For clients connecting to a SAS 9.2 or later version of the OLAP server, the following information is also available:

Last Update Time
indicates the last time the result set was accessed. For example, a result set is considered to be accessed when a method call such as ReadCells is made against it.

Query ID
is the query ID.

Read Cells
indicates the number of cells read.

Total Cells
indicates the total number of cells.

Note: For SAS versions earlier than 9.2, this item is equivalent to `Size`.

Here is an output example for LIST QUERIES when connected to an OLAP server prior to SAS 9.2:

```
Query Statement "FROM MDDBCARS SELECT"
Query ID: A170866D-2922-48B1-8CD8-A3BF2C97705D
Query type is Multidimensional
Query size in cells =  1
```

Here is an output example for LIST QUERIES when connected to a SAS 9.2 or later OLAP server:

```
Query Statement "FROM MDDBCARS SELECT"
Query ID: A170866D-2922-48B1-8CD8-A3BF2C97705D
Query type is Multidimensional
Query size in bytes =  685772
Total cells = 1 and cells read = 1
The time of last data set access is 12Apr2011:11:21:21

Query Statement "FROM SALES SELECT"
Query ID: 67F072EF-22B6-4EC9-B79A-013B72A5BE53
Query type is Flattened
Query size in bytes =  684244
Total cells = 1 and cells read = 1
The time of last data set access is 12Apr2011:13:55:04
```

Listing Session Rowsets on an OLAP Server

You can list information about session queries with the LIST ROWSETS statement.

If the SESSION option is specified, then the rowsets associated with a particular user are listed. Otherwise, all rowsets are listed. If the OUT option is specified, the output is written to the specified data set. The LIST ROWSETS statement is available only for SAS 9.2 or later OLAP servers. It is not available for OLAP servers prior to SAS 9.2.

LIST ROWSETS [SESSION=*owner*] [OUT=*data-set*] [NODEONLY];

The LIST ROWSETS statement generates the following information:

Cube names
lists the names of the cubes opened in this rowset.

Last Update Time
indicates the last time the rowset was accessed. For example, a rowset is considered to be accessed when a method call such as ReadRows is made against it.

Number of open cubes
indicates the number of cubes opened in this rowset.

Rowset ID
is the rowset ID.

Rowset name
indicates the type name of the rowset. The following are possible values for Rowset name.

- ACTIONSET
- CATALOGSET
- CUBESET
- DIMENSIONSET
- FUNCTIONSET
- HIERARCHYSET
- LEVELSET
- MEMBERSET
- MEASURESET
- PROPERTYSET
- SCHEMASET
- SETSET

Here is an output example for LIST ROWSETS (this command is available only when connected to a SAS 9.2 or later OLAP server):

```
Rowset name "CUBESET"
Rowset ID: 9793404A-D046-4CF0-A7ED-EE0F1BB643A6
The time of last rowset access is 12Apr2011:11:20:51

Rowset name "DIMENSIONSET"
Rowset ID: 66D02B81-2F92-464F-8782-7516E67493F4
The time of last rowset access is 12Apr2011:11:20:54
Cube CENSUSCMR is being accessed by this rowset.
```

Listing Session Cubes on an OLAP Server

You can list information about session cubes using the LIST CUBES statement.

LIST CUBES [CUBE=*cube-name*] [OUT=*data-set*];

If the CUBE option is specified, then only information about the specified cube is listed. Otherwise, all cubes are listed. If the OUT option is specified, the output is written to the specified data set.

The LIST CUBES statement generates the following information:

Cube name
is the name of the cube.

Cube status
indicates the status of the cube (enabled or disabled).

Number of open result sets
lists the number of open result sets using this cube.

Note: The LIST CUBES command is available only for clients connecting to a SAS 9.2 or later version of the OLAP server.

Closing a Session

You can close an OLAP session using the CLOSE SESSION statement.

CLOSE SESSION *id* | _ALL_ [USER=*user-name*] [CUBE=*cube-name*] [INACTIVE=*seconds*];

The ID is the unique session identifier returned from the LIST SESSIONS statement. The session is closed on the server. If the _ALL_ keyword is specified, then all open sessions are closed. If the USER option is specified, then only sessions owned by the specified user are closed. If the CUBE option is specified, then only sessions that have the specified cube open are closed. If the INACTIVE option is specified, then only sessions inactive for the specified seconds are closed.

Canceling a Query Result Set

You can cancel (close) an OLAP query result set with the CANCEL QUERY statement.

CANCEL QUERY *id* | _ALL_ [INACTIVE=*seconds*];

The ID is the unique result set identifier returned from the LIST QUERIES statement. Canceling a query stops the query processing and closes the result set on the server. If the _ALL_ keyword is specified, then all open query result sets are closed. If the INACTIVE option is specified, then only queries inactive for the specified duration are canceled.

Disabling a Cube

You can disable a cube by using the DISABLE CUBE statement. When you disable a cube, the server does not accept any queries against that cube. In addition, the cube does not show up in the list of available cubes in the client session.

DISABLE CUBE *cube-name* | _ALL_;

The statement disallows new MDX queries that reference the specified cube (or all cubes if _ALL_ is specified). It will terminate currently executing queries.

Enabling a Cube

You can enable access to a disabled cube by using the ENABLE CUBE statement. The statement allows queries to be run against the cube again.

ENABLE CUBE *cube-name* | _ALL_;

If the _ALL_ keyword is specified, all cubes are enabled for access by OLAP sessions.

Enabling and Disabling Cubes for a Cube Update

After a cube has been updated, the updates can be seen only after all open queries have finished using the cube. After that, the next open query displays the new updates. In a busy environment, with numerous sessions, this could take considerable time. Executing the DISABLE and then ENABLE statements forces new queries to immediately access the new data. In addition, if a cube is actively being queried, it must be disabled before you can use the DELETE or DELETE_PHYSICAL PROC OLAP options. For more information about updating cubes, see “Overview” on page 132 and the ADD_DATA PROC OLAP option.

Refreshing a Cube

You can refresh a cube on an OLAP server by using the REFRESH CUBE statement.

REFRESH CUBE *cube-name* | _ALL_;

The statement allows the server to access global calculated members and named sets that were defined or deleted since the cube header was loaded into the servers cube cache. This includes removing access to any global definitions that have been deleted since the server was started. This is particularly important for global members and sets created using the PROC OLAP DEFINE command, or those deleted using the PROC OLAP UNDEFINE command. If the _ALL_ keyword is specified, all cubes on the OLAP server are refreshed.

Note: The REFRESH CUBE statement does not show changes to a cube that were made with the UPDATE_IN_PLACE or the ADD_DATA PROC OLAP option. You can disable and then enable a cube to force a cube update so that it can be seen.

Stopping an OLAP Server

You can stop a running OLAP server by using the STOP SERVER statement.

STOP SERVER;

The STOP SERVER statement provides a clean shutdown of the current OLAP server. After the STOP SERVER statement is complete, the only statement that can be executed is a CONNECT statement to connect to another server.

Quiescing an OLAP Server

You can quiesce a running OLAP server by using the QUIESCE SERVER statement.

QUIESCE SERVER;

The QUIESCE SERVER statement provides a deferred shutdown of the current OLAP server. After all open sessions have been closed, the OLAP server will shut down. Other OLAPOPERATE commands can be executed until the last open session is closed and the OLAP server stops. After the OLAP server stops, the only command that can be executed is a CONNECT command to connect to another server.

Pausing an OLAP Server

You can pause a running OLAP server by using the PAUSE SERVER statement.

PAUSE SERVER;

The PAUSE SERVER statement provides a way to disallow non-administrative operations on the current OLAP server. After the PAUSE SERVER command is complete, normal OLAP server operations are restored by submitting the RESUME SERVER command.

Resuming an OLAP Server

You can resume operation of a paused OLAP server by using the RESUME SERVER statement.

RESUME SERVER;

The RESUME SERVER statement provides a way to resume normal operations on a paused OLAP server. If this command is submitted on an OLAP server that is not paused, then it will execute normally, but have no affect on the OLAP server.

Stopping an OLAP Server Cluster

STOP CLUSTER;

The STOP CLUSTER statement enables you to perform a clean shutdown of the current OLAP server load-balanced cluster. After the STOP CLUSTER command is complete, the only command that can be executed is a CONNECT command to connect to another server.

Quiescing an OLAP Server Cluster

QUIESCE CLUSTER;

The QUIESCE CLUSTER statement enables you to defer a shutdown of the current OLAP server load-balanced cluster. After all open sessions have been closed, the OLAP server load-balanced cluster will shut down. Other OLAPOPERATE commands can be executed until the last open session is closed and the OLAP server load-balanced cluster stops. After the OLAP server load-balanced cluster stops, only the CONNECT command can be executed in order to connect to another server.

Pausing an OLAP Server Cluster

PAUSE CLUSTER;

The PAUSE CLUSTER statement enables you to disallow non-administrative operations on the current OLAP server load-balanced cluster. After the PAUSE CLUSTER command is complete, normal OLAP server load-balanced cluster operations are restored by submitting the RESUME CLUSTER command.

Resuming an OLAP Server Cluster

RESUME CLUSTER;

The RESUME CLUSTER statement enables you to resume normal operations on a paused OLAP server load-balanced cluster. If this command is submitted on an OLAP server load-balanced cluster that is not paused, it will execute normally. However, it will not have an effect on the OLAP server load-balanced cluster.

Chapter 14

OLAPCONTENTS Procedure

Overview: OLAPCONTENTS Procedure

What Does the OLAPCONTENTS Procedure Do?

The OLAPCONTENTS procedure enables you to connect to a SAS OLAP server and generate reports for SAS OLAP cubes. These reports list structural information for the cube, including details about the tables that were used to load the cube, the dimension structures (hierarchy, level, property), and measures. The OLAPCONTENTS procedure is similar to the CONTENTS procedure for SAS data sets. The DISPLAY CUBE command specifies the cube that you are generating a report for and any reporting options.

Syntax: OLAPCONTENTS Procedure

Example: `PROC OLAPCONTENTS [<connection-options>];`

```
   CONNECT <connection-options>;
   DISPLAY CUBE cube-name [OUT=data-set]<display-options>;
DISCONNECT;
RUN;
```

PROC OLAPCONTENTS [<*connection-options*>];
CONNECT <*connection options*>;
DISPLAY CUBE *cube-name* [OUT=*data-set*] <*display-options*>;
DISCONNECT;
RUN;

Usage: OLAPCONTENTS Procedure

Connecting to an OLAP Server

The CONNECT command enables you to establish a connection to a running SAS OLAP Server. This connection enables you to obtain cube information that is used to generate the reports that OLAPCONTENTS provides. You can have only one connection to an OLAP server at a time.

CONNECT <*connection options*>;

Here are the options for the CONNECT command:

HOST=
specifies the host name or network IP address of the computer hosting the SAS OLAP Server (for example, HOST=d1234.na.sas.com). The value `localhost` can be used if the SAS session is connecting to a server on the same computer.

PASSWORD= |
specifies the password that corresponds to the user ID.

PORT=
specifies the TCP port to which the SAS OLAP Server listens for connections (for example PORT= 5451).

Note: COM connection support was discontinued in SAS 9.3.

PROTOCOL=
specifies the network protocol for communicating with the SAS OLAP Server. The default value is BRIDGE.

Note: COM connection support was discontinued in SAS 9.3.

SSPI
(Security Support Provider Interface) specifies that the identity of the user running the SAS session will be used to establish the connection. The USERID

and PASSWORD options are unnecessary and are ignored when SSPI is specified.

Restriction This connection option is used only when connecting to an OLAP server running on a Windows machine.

SPN=
specifies the Service Principal Name of the connection destination. This option is optional even if SSPI is specified. When SPN is not specified, one is created using the HOST= and PORT= values.

Restriction This connection option is used only when connecting to an OLAP Server running on a Windows machine.

USERID=
specifies a user ID to be used to connect to the SAS OLAP server.

Displaying Cube Information

After a connection to an OLAP server is made, you can generate and display cube information using the DISPLAY CUBE command. Here is the syntax for the command:

DISPLAY CUBE *cube-name* [OUT=*data-set*] <*display-options*>;

You can issue multiple DISPLAY CUBE statements for cubes during the same connection. The information that is generated for the cube is structural in nature and includes details about the tables that were used to load the cube, dimension structures (hierarchy, level, property), and measures. By default, output is sent to an output window, by way of ODS. If the OUT= option is specified, then the output is written to the specified data set.

The default cube report that is generated can be lengthy, depending on the cube that you are working with. If you need to focus on a specific area of cube information, you can refine the report that is generated and focus on specific areas of cube information. The DISPLAY CUBE command has several options that enable you to filter out part of the information that is generated. When these options are set, specific cube information is disabled and the report that is generated is reduced.

Here are the options for the DISPLAY CUBE command:

NODIM | NODIMENSION
specifies that the dimension information will be filtered from the DISPLAY CUBE output.

NOHIER | NOHIERARCHY
specifies that the hierarchy information will be filtered from the DISPLAY CUBE output.

NOLEV | NOLEVEL
specifies that the dimension level information will be filtered from the DISPLAY CUBE output.

NOPROP | NOPROPERTY
specifies that the property information will be filtered from the DISPLAY CUBE output.

NOCALC | NOCALCULATED
specifies that the calculated member information will be filtered from the DISPLAY CUBE output.

NOMEAS | NOMEASURE
specifies that the measure information will be filtered from the DISPLAY CUBE output.

NOSET
specifies that the named set information will be filtered from the DISPLAY CUBE output.

OUT=
specifies a data set that the DISPLAY CUBE output is sent to. By default, the generated report is sent to an output window, by way of ODS. However, you can direct the output for the report to a data set using the OUT= option on the DISPLAY CUBES command.

SUMMARYONLY
specifies that only the summary information will be displayed in the DISPLAY CUBE output.

Disconnecting from an OLAP Server

You can disconnect from the current SAS OLAP server with the DISCONNECT statement. After you have disconnected from the server, you can reconnect to that or another OLAP server with the CONNECT statement.

It is possible to terminate the OLAPOPERATE procedure with the QUIT or RUN statements. With either of these statements, an automatic server disconnection is executed. It is not necessary to explicitly disconnect from the server before terminating the procedure.

DISCONNECT;

OLAPCONTENTS Examples

The following output is an example of what would be displayed when the cube SIMBA is specified:

```
The SAS System

                              The OLAPCONTENTS Procedure

Cube Name                SIMBA                     Dimensions          4
Cube Description         This is the SIMBA Cube    Hierarchies         5
Cube Fact Table          TOLAPSIO.FACTTABC         Levels              15
Cube Drillthrough Table  OLAPSIO.SIMBA             Properties          13
                                                   Calculated Members  1
                                                   Measures            13
                                                   Named Sets          1
```

List of DIMENSIONS

Dimension Name	Dimension Type	Number of Hierarchies	Dimension Table	Dimension Caption	Dimension Description
MARKET	STANDARD	2	OLAPSIO.MARKET	MARKET	MARKET
PRODUCT	STANDARD	1	OLAPSIO. PRODUCT	PRODUCT	this is the complete product description
SUPPLIER	STANDARD	1	OLAPSIO.SUPPLY	SUPPLIER	SUPPLIER
TIME	TIME	1		TIME	The TIME description

DIMENSION

Dimension Name	Dimension Type	Number of Hierarchies	Dimension Table	Dimension Caption	Dimension Description
MARKET	STANDARD	2	OLAPSIO.MARKET	MARKET	MARKET

HIERARCHY for MARKET Dimension

Hierarchy Name	Number of Levels	Number of Properties	Number of Calculated Members	Hierarchy Caption	Hierarchy Description
GEOGRAPHICAL	3	2	0	GEOGRAPHICAL	this is the geographical description

LEVELS for GEOGRAPHICAL Hierarchy

Level	Level Name	Member Cardinality	Level Caption	Level Description
1	All	1	All GEOGRAPHICAL	All GEOGRAPHICAL
2	REGION	4	My REGION level caption	My REGION level caption
3	STATE	20	STATE	STATE

PROPERTIES for GEOGRAPHICAL Hierarchy

Property Name	Property Caption	Property Description	Property Owning Level

```
       REGIONALDIRECTOR    REGIONALDIRECTOR         REGIONALDIRECTOR
REGION
       REGIONSIZE          My REGIONSIZE caption    My REGIONSIZE caption
REGION


                                HIERARCHY for MARKET Dimension

                Number                Number of
  Hierarchy         of   Number of  Calculated  Hierarchy
  Name          Levels  Properties     Members  Caption       Hierarchy Description

  POPULATION         3           0           0  POPULATION
POPULATION


                                LEVELS for POPULATION Hierarchy

                                   Member
   Level    Level Name         Cardinality    Level Caption             Level
Description

       1    All                          1    All POPULATION            All
POPULATION
       2    POPULATIONGROUP              3    POPULATIONGROUP
POPULATIONGROUP
       3    STATE                       20    STATE
STATE


                                         DIMENSION

Dimension                    Number of Dimension      Dimension
Name      Dimension Type   Hierarchies Table          Caption   Dimension Description

PRODUCT   STANDARD                   1 OLAPSIO.       PRODUCT   this is the
complete
                                       PRODUCT                  product
description


                                HIERARCHY for PRODUCT Dimension

                Number                Number of
  Hierarchy         of   Number of  Calculated  Hierarchy
  Name          Levels  Properties     Members  Caption       Hierarchy Description

  PRODUCT            3           3           0  PRODUCT
PRODUCT


                                LEVELS for PRODUCT Hierarchy

                                   Member
   Level    Level Name         Cardinality    Level Caption             Level
Description
```

```
       1    All                          1    All PRODUCT               All
PRODUCT
       2    FAMILY                       4    FAMILY
FAMILY
       3    SKU                         13    SKU
SKU
```

PROPERTIES for PRODUCT Hierarchy

```
                                                                          Property
Owning
       Property Name       Property Caption         Property Description     Level

       FAMILYCODE          FAMILYCODE               FAMILYCODE
FAMILY
       FAMILYINTRODATE     FAMILYINTRODATE          FAMILYINTRODATE
FAMILY
       SKUCODE             SKUCODE                  SKUCODE
SKU
```

DIMENSION

```
Dimension                    Number of Dimension      Dimension
Name      Dimension Type   Hierarchies Table          Caption   Dimension Description

SUPPLIER  STANDARD                  1 OLAPSIO.SUPPLY SUPPLIER
SUPPLIER
```

HIERARCHY for SUPPLIER Dimension

```
                Number                Number of
  Hierarchy         of   Number of  Calculated  Hierarchy
  Name          Levels  Properties     Members  Caption       Hierarchy Description

  SUPPLIER           2           5           0  SUPPLIER
SUPPLIER
```

LEVELS for SUPPLIER Hierarchy

```
                                    Member
   Level    Level Name          Cardinality    Level Caption             Level
Description

       1    All                          1    All SUPPLIER              All
SUPPLIER
       2    SUPPLIERNAME                 3    SUPPLIERNAME
SUPPLIERNAME
```

PROPERTIES for SUPPLIER Hierarchy

Property Name	Property Caption	Property Description	Property Owning Level
ADDRESS	ADDRESS	ADDRESS	SUPPLIERNAME
CITY	CITY	CITY	SUPPLIERNAME
COUNTRY	COUNTRY	COUNTRY	SUPPLIERNAME
SSTATE	SSTATE	SSTATE	SUPPLIERNAME
ZIP	ZIP	ZIP	SUPPLIERNAME

DIMENSION

Dimension Name	Dimension Type	Number of Dimension Hierarchies	Dimension Table	Dimension Caption	Dimension Description
TIME	TIME	1		TIME	The TIME description

HIERARCHY for TIME Dimension

Hierarchy Name	Number of Levels	Number of Properties	Number of Calculated Members	Hierarchy Caption	Hierarchy Description
TIME	4	3	1	TIME	TIME

LEVELS for TIME Hierarchy

Level	Level Name	Member Cardinality	Level Caption	Level Description
1	All	1	All TIME	All TIME
2	YEAR	2	My YEAR caption	My YEAR caption
3	QUARTER	4	My QUARTER caption	My QUARTER caption
4	MONTH	12	My MONTH caption	My MONTH caption

PROPERTIES for TIME Hierarchy

Property Name	Property Caption	Property Description	Property Owning Level

```
       SAS_MEMBER_DATE     SAS_MEMBER_DATE          SAS_MEMBER_DATE
MONTH
       SAS_MEMBER_DATE     SAS_MEMBER_DATE          SAS_MEMBER_DATE
QUARTER
       SAS_MEMBER_DATE     SAS_MEMBER_DATE          SAS_MEMBER_DATE
YEAR
```

CALCULATED MEMBERS for TIME Hierarchy

```
Member Name  Member Unique Name   Member Expression

june         [time].[all
6
             time].
[june]
```

List of MEASURES

```
 Measure
 Name      Measure Statistic    Measure Caption   Measure Description Measure Expression

 COGS      SUM                  Sum of COGS       Cost of Goods
Sold
 good      CALCULATION          good              good                aggregate([time].
[all
                                                                      time].
[2000].children,
                                                                      [measures].
[profit])
 margin    CALCULATION          margin            margin              [measures].[sales]
-
                                                                      [measures].
[cogs],format_

string='dollar16.2'
 MARGIN_   CALCULATION          MARGIN_PERCENT    MARGIN_PERCENT      ([measures].
[margin] /
 PERCENT                                                              [measures].
[sales]) * 100,

format_string='8.2'
 MARKET_   SUM                  Sum of            Marketing
Expense
 EXP
MARKETINGEXPENSE
 MISC_     SUM                  Sum of            Misc
Expense
 EXPENSE
MISCEXPENSE
 nu_region                      Number of Unique  Number of
Unique
                                Values for region Values for
region
```

```
                              in GEOGRAPHICAL   in
GEOGRAPHICAL
 PAY_EXP    SUM               Sum of            Payroll
Expense

PAYROLLEXPENSE
 PROFIT     CALCULATION       PROFIT            PROFIT             [measures].
[margin] -
                                                                  [measures].
[total_expenses],

format_string='dollar20.2'
 PROFIT_    CALCULATION       PROFIT_PERCENT    PROFIT_PERCENT     ([measures].
[profit] /
 PERCENT                                                          [measures].
[sales]) * 100,

format_string='8.2'
 SALES      SUM               Sum of SALES      Sum of
SALES
 Total      CALCULATION       Total Sales       Total Sales
Aggregate([measures].[SALES])
 Sales                                                            ,
GROUP='[Year To Date]'
 TOTAL_     CALCULATION       TOTAL_EXPENSES    TOTAL_EXPENSES     [measures].
[market_exp] +
 EXPENSES                                                         [measures].
[pay_exp] +
                                                                  [measures].
[misc_expense],

format_string='dollar20.2'


                                List of NAMED SETS

              Set
              Name     Set Expression

              myset    {[measures].[margin], [measures].[profit],[measures].[good]}
```