Paper 104-29

Designing Information Maps for Ad Hoc Reporting

Diane Hatcher, SAS, Cary, NC

ABSTRACT

SAS® information maps provide a layer of metadata that describes your physical data structures in terms that business users can understand. In the SAS® 9 Business Intelligence reporting interfaces, business users are presented with a single list of items they can use to answer their questions. Based upon the user's selection, SAS will automatically generate the appropriate query code. This paper will provide helpful tips on how to create information maps to provide your business users with the appropriate items to ensure accurate and consistent results. We will explore how certain settings in information maps can impact SQL and MDX code generation. This paper is written for people whose role is to provide access to enterprise data for business users.

INTRODUCTION

Once you have registered your physical data sources in the SAS® Metadata Server, you can define SAS information maps – SAS' business metadata layer that describes your physical data warehouse in easy-to-understand terms. This business metadata layer can enable business users to be self-sufficient in asking questions from the data without having advanced technical know-how or going through extensive training. This metadata layer captures three key types of information:

- **Customized labels and descriptions that are understandable by the target business users**
  Business users need to have a strong understanding of the data they are querying in order to ask the right questions. It is critical that they understand what data elements they are requesting. Information maps allow you to supply descriptive labels and detailed descriptions that business users can understand. By utilizing customized information maps for different user groups, you can ensure that the data is surfaced in context to the way each group works.

- **Business rules that drive consistent delivery of information across the organization**
  Consistency of information is critical for enterprise-wide deployment of a data warehouse. Because the possible queries are so diverse, you must establish boundaries for how the data is accessed. Information maps capture several types of business rules—standard calculations, standard filters, and allowable actions—that help to ensure that the information generated is consistent and accurate.

- **Details about the data sources so SAS can automatically generate the appropriate query code**
  Much of the information captured in an information map is used by SAS to generate appropriate query code automatically. Since most business users are not programmers, they can not be expected to build their own queries. Once the business user has selected data items from the information map, SAS Query Services reads the relevant metadata and generates the appropriate query for them.

SAS® Information Map Studio (IMS) is the desktop Java application for building and managing information maps. IMS is targeted at information architects or business data modelers. This persona (I'll call him Marcel) is someone who has strong experience with SQL or MDX, knows the data available, and works closely with the business units to understand the questions they need to ask. Marcel will utilize IMS to build information maps that can be accessed by his business users through SAS reporting applications like SAS® Web Report Studio or SAS® Information Delivery Portal.

This paper highlights some of the tips and techniques that you can utilize in setting up an information map to ensure that the SQL or MDX code that SAS generates is what he expects. Uncovering some of the rules that SAS utilizes in generating code will help you make better decisions on how to setup data items to deliver the expected results.

TIPS AND TECHNIQUES FOR HANDLING RELATIONAL DATA

Relational data refers to data sources that consist of 2-dimensional tables that can be joined together and accessed using SQL code. Common relational database systems include SAS datasets, SAS/SPDS, Teradata, Oracle, DB2, SQL Server, local PC files, and others. SAS (and information maps) can access third party databases if the appropriate SAS/ACCESS® engines are being utilized.

Even though IMS provides dialogs to assist in creating data items, there are rules that SAS uses to generate appropriate query code. This section discusses three of the areas that impact how to setup your information map for relational data:

- Building expressions for new measures
- Building filters with category data items versus measure data items
- Dealing with potential SQL traps

A data item is an element defined in the information map that refers back to the physical data. It can refer to a single column from a physical table or an expression that utilizes multiple columns. A data item is also classified as either a category item or a measure item.

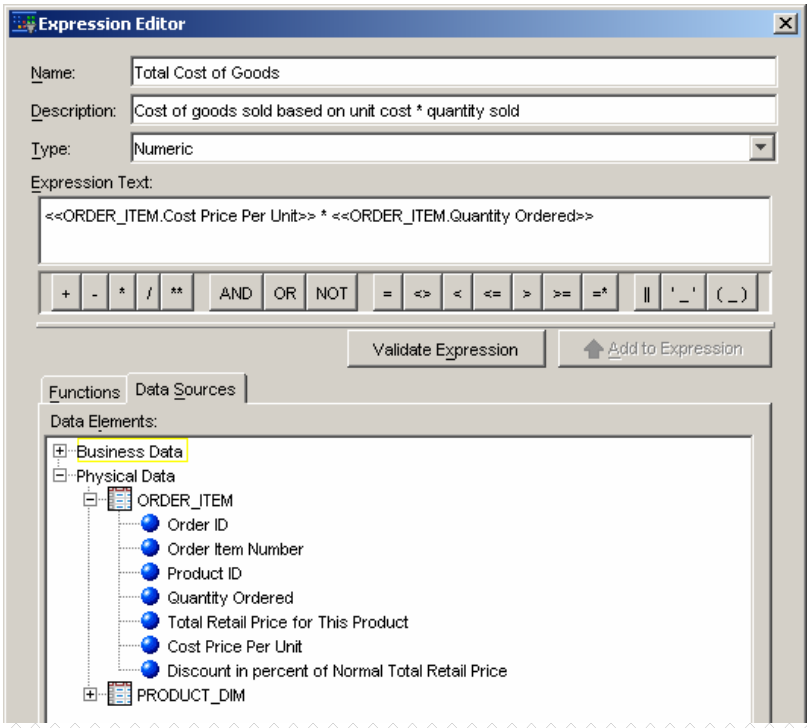
- Category data items are those that are used to aggregate data—essentially variables in relational data that are included in a "GROUP BY" statement in SQL.
- Measure data items are those that are aggregated—variables that are used with aggregate functions such as "SUM(_)". Measure data items are always associated with a default aggregate function.

BUILDING EXPRESSIONS FOR NEW MEASURES

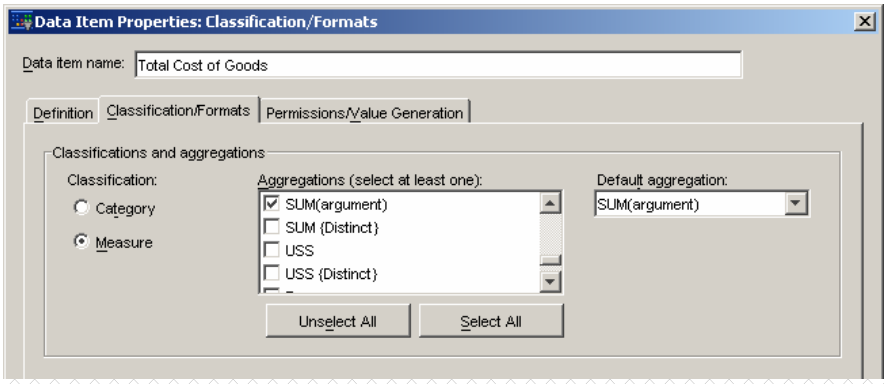
As mentioned earlier, information maps can contain calculations that describe values utilized by the business unit. For example, the data source may only contain variables for `Unit Cost` and `Quantity Sold`, but the business user may also want to also know `Total Cost`. A measure data item can be added to the information map that defines `Total Cost = Unit Cost * Quantity Sold`. More complex expressions are also supported, providing you with the opportunity to centrally define an expression that can be used by all your business users.

Using physical columns versus data items

When building a calculation, you have the option of creating a measure with physical column references OR data items. You can not mix both physical columns and data items in the same expression. In the example below, I have used physical columns in the expression.



Since this will be a measure data item, you need to associate it with a default aggregate function. In the properties dialog, you can set the default aggregation. In this case, I've set it to SUM(_).

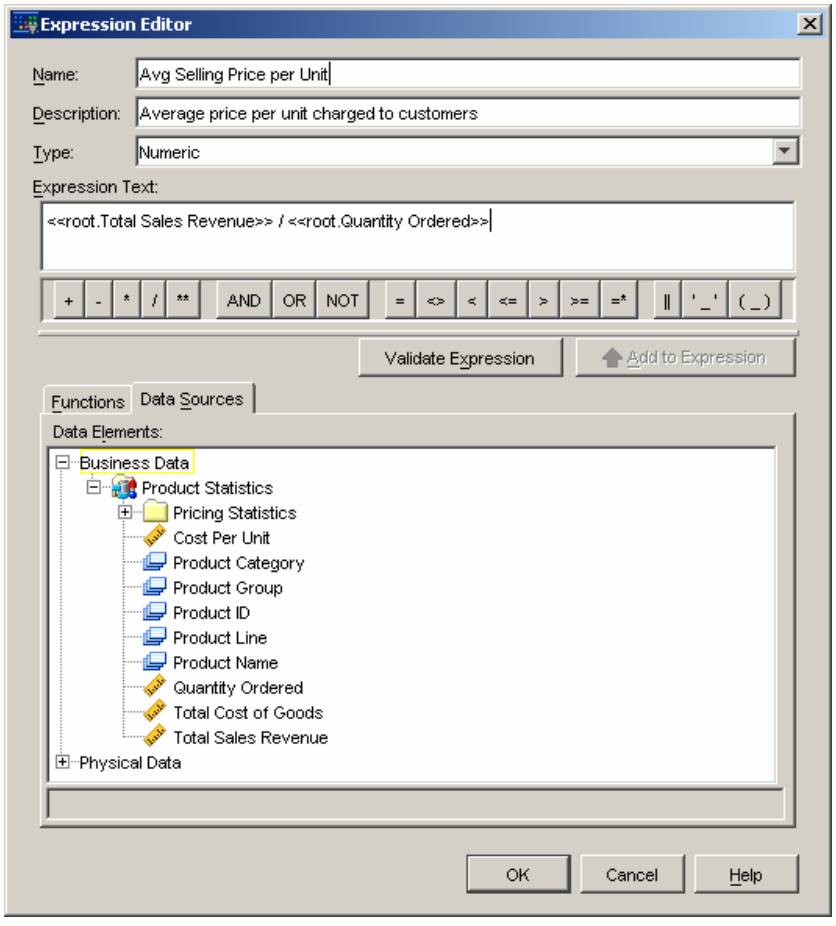


The SQL code that is generated when the "Total Cost of Goods" data item is selected is:

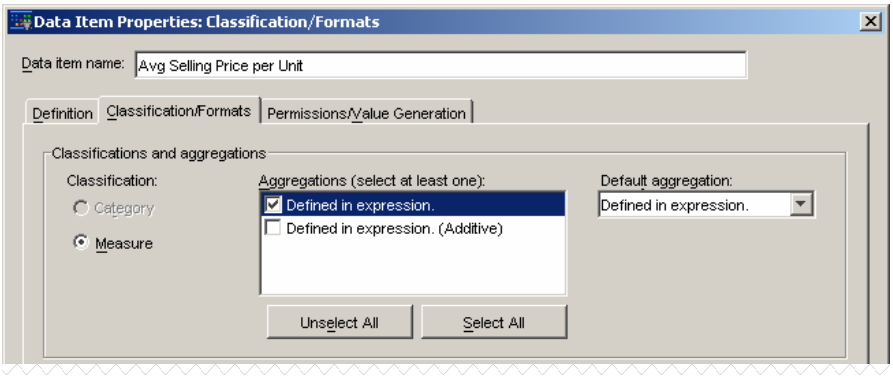
```
SUM( ( table1.CostPrice_Per_Unit * table1.Quantity ) )
```

In this case, the aggregate function is wrapped around the entire expression. Using physical columns in the expression is best for measures that will be summarized as sub-totals or totals.

When using data items in the expression, the SQL code will be slightly different. Below is an expression that utilizes data items, instead of physical columns.



When using data items, there is no need to establish a default aggregate function. Since each business item is already associated with an aggregate function, the SQL will be generated using those settings. Looking at the properties tab, you can see that the aggregate function is set to “Defined in expression”.



The SQL code that is generated when “Avg Selling Price per Unit” data item is selected is:

```
( SUM(table0.Total_Retail_Price) / SUM(table0.Quantity) )
```

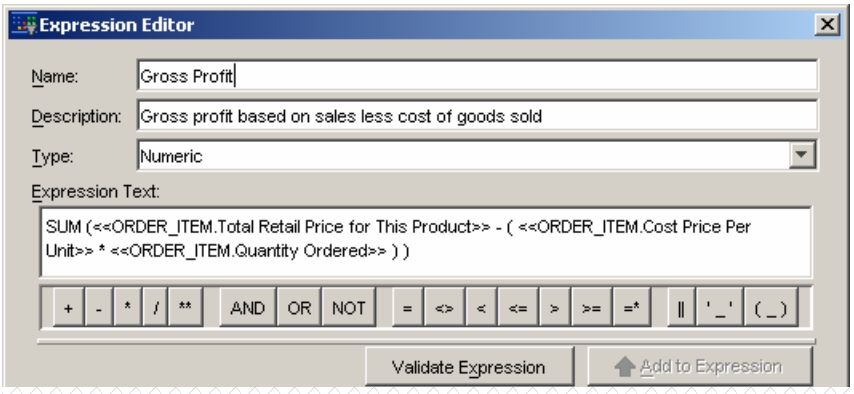
In this case, each data item has a separate aggregate function. Using data items in the expression is best for measures that need to be calculated as weighted averages—such as Average Unit Price.

Defining complete expressions manually

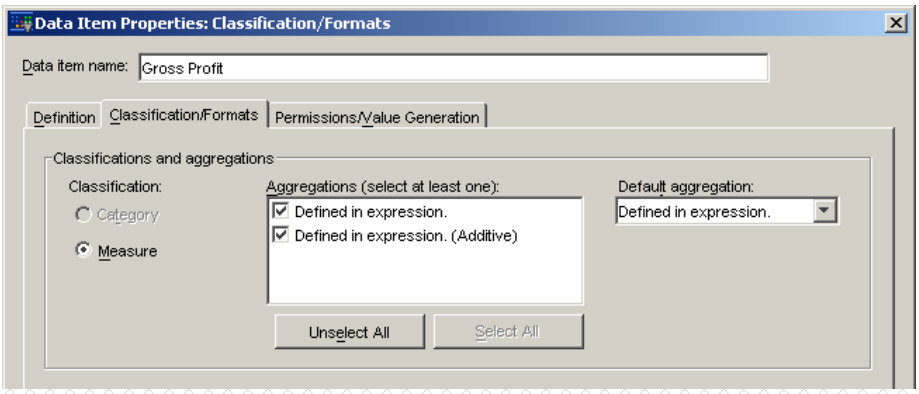
You can manually build an expression by typing directly in the “Expression Text” area. This allows you to re-use existing code, build complex expressions, or utilize database-specific functions not available on the Functions tab provided in the IMS interface. General syntax rules for building an expression are:

- Enclose each variable with << >>
- Preface physical column references with the table name and separate with a “.”, such as `table.column`

A **complete** expression also includes any aggregate functions.



Since we have built a complete expression (with the aggregate function included), the aggregations are flagged as being defined in the expression. This setting is automatically determined by the application based on the expression.



The SQL that is generated when "Gross Profit" data item is selected is

```
( SUM (table0.Total_Retail_Price - ( table0.CostPrice_Per_Unit * table0.Quantity ) ) )
```

SAS Query Services converts the column labels to the actual physical column names when generating the query.

BUILDING FILTERS WITH CATEGORY DATA ITEMS VERSUS MEASURE DATA ITEMS

Another type of business rule that information maps capture are filter items. Filter items are expressions that are translated into WHERE or HAVING clauses in SQL code. WHERE clauses are evaluated for individual records (prior to any aggregations), and HAVING clauses are evaluated on summarized information.

When building a filter in IMS, you must select data items for the expression. You can not use a physical column directly in the filter expression. You can use either category data items or measure data items in the expression.

- Category data items will generate a WHERE clause (filter is evaluated prior to any aggregations)
- Measure data items will generate a HAVING clause (filter is evaluated after aggregating the data)

For example, your business users need to report on individual products that are profitable. The data source includes multiple sales transactions for a given product. Therefore, profits need to be summarized for each product to determine which ones are profitable. In this case, you can build a filter using the "Gross Profit" measure data item, which will generate the following SQL code:

```
SELECT table1.Product_Name AS DIR_1 LABEL='Product Name',
( SUM (table0.Total_Retail_Price - ( table0.CostPrice_Per_Unit * table0.Quantity ) ) )  AS
       DIR_2 LABEL='Gross Profit'
FROM ordetail.ORDER_ITEM table0 Inner join orstar.PRODUCT_DIM table1 on table0.Product_ID =
       table1.Product_ID
GROUP BY DIR_1
HAVING DIR_2 >= 0
```

As a result of the HAVING clause, Gross Profit will be summarized for each Product, and then evaluated against the filter.

However, if your business users need to evaluate a measure data item on a per-record basis, you will need to create a separate category data item referencing the same physical data. For example, your business users also need to summarize profitable sales transactions by product. Using "Gross Profit" data item in the filter will not generate the desired results. In this case, you need to create a category data item that also refers to Gross Profit. Let's call it "Gross Profit per Transaction".

Using “Gross Profit per Transaction” data item in the filter will generate the following SQL code:

```
SELECT table1.Product_Name AS DIR_1 LABEL='Product Name',
( SUM (table0.Total_Retail_Price - ( table0.CostPrice_Per_Unit * table0.Quantity ) ) )  AS
       DIR_2 LABEL='Gross Profit'
FROM ordetail.ORDER_ITEM table0 Inner join orstar.PRODUCT_DIM table1 on table0.Product_ID =
       table1.Product_ID
WHERE ( table0.Total_Retail_Price - (table0.CostPrice_Per_Unit * table0.Quantity) )  >= 0
GROUP BY DIR_1
```

The WHERE clause will evaluate the filter against each sales transaction in the data source, and then summarize the results by product.

DEALING WITH POTENTIAL SQL TRAPS

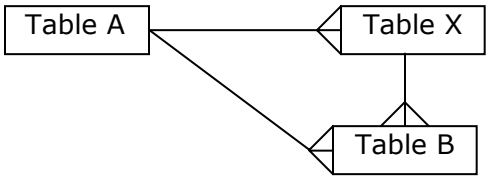
In relational data warehouses, join paths between tables could cause problems for SQL code generation. This section will briefly discuss three common scenarios and provide some basic guidelines on how to deal with them in this release of IMS.

- Multiple join path options
- Aggregating measures from both the parent and child tables
- Many-to-many joins

Multiple join path options

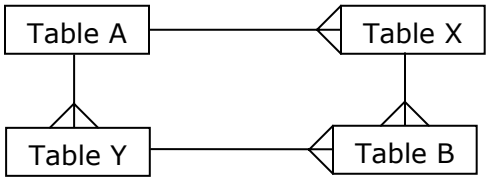
When columns are being requested from two separate tables, the tables must be related to each through a join path. In the simplest situation, they are joined directly to each other. However, there are situations where there are multiple join paths connecting the tables.

Situation 1: Multiple join paths between Table A and Table B, where one path is more direct



In this situation, SAS Query Services will utilize the more direct join path—i.e., joining Table A directly to Table B. You can not specify to use the less direct path for query generation in this version of IMS, unless you remove the direct join relationship.

Situation 2: Multiple join paths between Table A and Table B, where all paths are equally direct

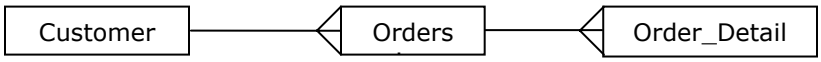


In this situation, SAS Query Services will select one of these join paths when generating SQL code. To make it clear which join path to utilize, you should only establish a single join path between Table A and Table B. If there is a business requirement to support both join paths for reporting purposes, you can resolve this situation by surfacing each join path in a separate context:

1) You could create a separate information map for each join path. By selecting a specific information map, business users will understand the context of the data they are accessing.
2) You could utilize table aliases to create a separate group of data items for each join path within the information map. By selecting certain data items (with appropriate labels), business users can select the data items for the context they need.

Aggregating measures from both the parent and child tables

In a join relationship where the cardinality is one-to-many, the child table contains multiple records for each record in the parent table. If you need to aggregate measures in the parent table simultaneously with measures in the child table, the results could be erroneous. For example, you have the following data structure:



To find the count of orders and order line items, you generate the following SQL code:

```
SELECT Customer.Customer_Account, COUNT(Orders.Order_ID),
       COUNT(Order_Detail.Order_Line_Number)
GROUP BY Customer.Customer_Account
```

The result would show that the number of Orders is the same as the number of Order Line Items. This is not correct as there should be fewer Orders than Order Line Items. For this particular scenario, here are three possible ways you can resolve this situation:

1) Utilize the COUNT(Distinct) aggregate function on the Orders variable.

```
SELECT Customer.Customer_Account, COUNT(DISTINCT Orders.Order_ID),
       COUNT(Order_Detail.Order_Line_Number)
GROUP BY Customer.Customer_Account
```

2) Utilize the child table to surface measures related to the parent record. This could require adding additional columns to the physical table.

```
SELECT Customer.Customer_Account, COUNT(DISTINCT Order_Detail.Order_ID),
       COUNT(Order_Detail.Order_Line_Number)
GROUP BY Customer.Customer_Account
```

These solutions above only work if the measure used with the DISTINCT option contains unique values. In the example above, the values for `Order_ID` are unique.

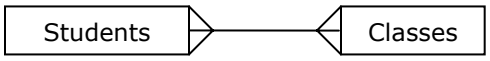
3) If the measure being aggregated from the parent table does not contain unique values, you can define an expression that creates unique values.

```
SUM(DISTINCT table0.Order_Amt + (0.0000000000001  *  table0.Order_ID))
```

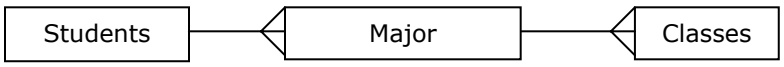
In this example, table0.ORDER_ID is a 10-digit unique integer value, so the 13 decimal places in the multiplier are needed to eliminate any potential rounding problems.

Many-to-many joins

Many-to-many joins are not currently supported in IMS. These situations have potential to generate inaccurate results, and in cases of high cardinality, very slow performance.



To solve this problem, you need to insert 1-to-many bridge tables that eliminate the need for many-to-many relationships.



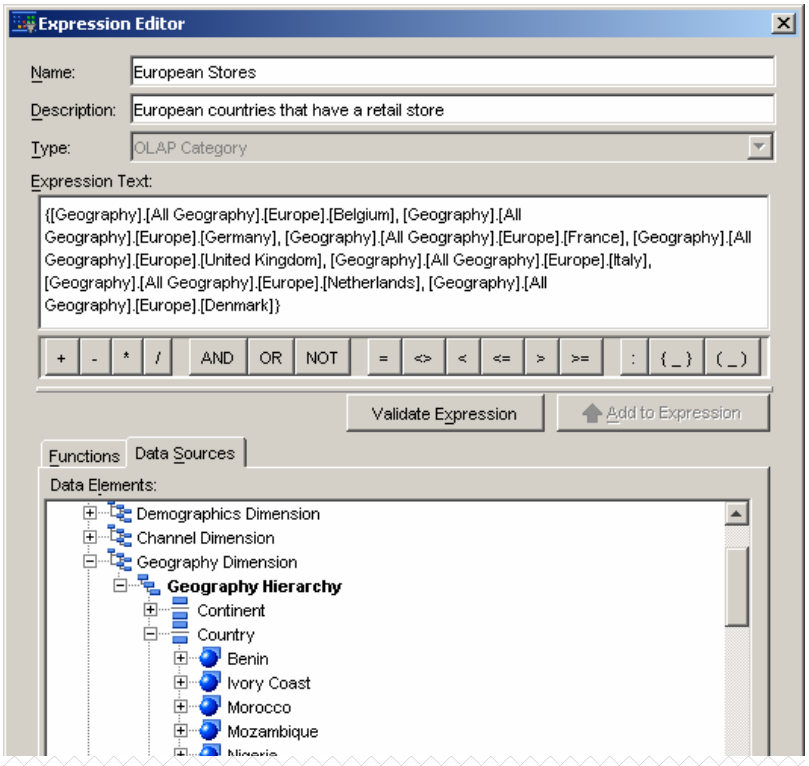
TIPS AND TECHNIQUES FOR HANDLING OLAP DATA

Information maps can also reference a SAS 9 OLAP data source. The same advantages that apply for relational data also apply to OLAP data. You can surface only the relevant hierarchies and measures from the cube and rename them with meaningful labels. You can also supplement the data in the cube with additional measures and member sets. Member sets are especially powerful, as they can be used to surface an initial view to the cube in downstream reporting applications like SAS Web Report Studio and SAS Information Delivery Portal.

DEFINING MEMBER SETS

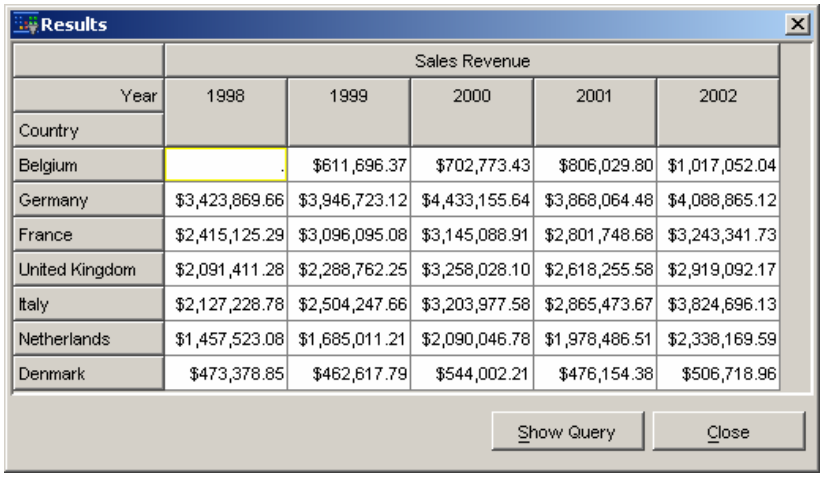
Member sets consist of a pre-defined list of member values from a single dimension in the cube, and they are classified as OLAP category data items. Category data items represent hierarchies—the navigational path through the cube. Measure data items represent measures that are physically part of the cube (both straight measures and defined measures) or additional calculated expressions defined in the information map. When the business user selects a category data item, he/she can then navigate (drill down, drill up, expand, etc.) through the cube. Category data items defined as member sets provide an initial view of the cube with only those items in the member set.

To create a member set, you must reference the physical cube metadata, as you need access to the member values. To create a member set, you must wrap the list of members within "{ _ }".



Here is the MDX code that is generated:

```
SET [&1European Stores] AS '( {[Geography].[All Geography].[Europe].[Belgium],
    [Geography].[All Geography].[Europe].[Germany], [Geography].[All
    Geography].[Europe].[France], [Geography].[All Geography].[Europe].[United Kingdom],
    [Geography].[All Geography].[Europe].[Italy], [Geography].[All
    Geography].[Europe].[Netherlands], [Geography].[All Geography].[Europe].[Denmark]} )'
```



	Sales Revenue				
Year / Country	1998	1999	2000	2001	2002
Belgium	.	$611,696.37	$702,773.43	$806,029.80	$1,017,052.04
Germany	$3,423,869.66	$3,946,723.12	$4,433,155.64	$3,868,064.48	$4,088,865.12
France	$2,415,125.29	$3,096,095.08	$3,145,088.91	$2,801,748.68	$3,243,341.73
United Kingdom	$2,091,411.28	$2,288,762.25	$3,258,028.10	$2,618,255.58	$2,919,092.17
Italy	$2,127,228.78	$2,504,247.66	$3,203,977.58	$2,865,473.67	$3,824,696.13
Netherlands	$1,457,523.08	$1,685,011.21	$2,090,046.78	$1,978,486.51	$2,338,169.59
Denmark	$473,378.85	$462,617.79	$544,002.21	$476,154.38	$506,718.96

CONCLUSION

SAS information maps allow you to create a reporting environment that enables your business users to be self-sufficient in asking questions of your data warehouse. Information maps provide you with the ability to:

- **Customize labels and descriptions to your business users**
- **Define business rules that drive consistent delivery of information across the organization**
- **Provide details about the data sources so SAS can automatically generate the appropriate query code**

Your business users will access information maps in SAS reporting applications like SAS® Web Report Studio and SAS® Information Delivery Portal. Information maps make the data source transparent, as business users see only a list of understandable data items. SAS generates the appropriate SQL or MDX code based upon the user's request, providing self-service querying capabilities to your business users while preserving consistency and accuracy of the results.

Summary of the key tips and techniques discussed:

- Using physical columns in the expression is best for measures that will be summarized as sub-totals or totals.
- Using data items in the expression is best for measures that need to be calculated as weighted averages.
- When manually creating complete expressions, enclose variables with << >>, utilize the format "`table.column`", and include the necessary aggregate function(s).
- For filters, use category data items to generate a WHERE clause, and use measure data items to generate a HAVING clause.
- Surface multiple join paths between two tables in separate information maps or as separate data items to provide the appropriate context to business users.
- Try not to allow aggregation of measures from both the parent and child tables in the same query. If this is necessary, the use of DISTINCT can be used with unique values.
- Redesign many-to-many joins to be a series of 1-to-many relationships.
- With an OLAP data source, build member sets to establish an initial view of a dimension for business users.

ACKNOWLEDGMENTS

Special thanks go to Julie Radford, SAS Usability Analyst, Jeff Diamond, SAS Development Manager for Java OLAP components, Mike Whitcher, SAS Development Manager for Query Services, Scott Marusak, SAS Development Manager for Information Map Studio, Jens Dahl Mikkelsen, SAS Development Product Manager, and Jeff Shaughnessy, SAS Senior Development Manager for BI Clients for providing input and editing support for the paper.

CONTACT INFORMATION

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

Author Name	Diane Hatcher, Worldwide Marketing Strategy
Company	SAS
Address	SAS Campus Drive
City state ZIP	Cary, NC, 27511
Work Phone:	(919) 531-0503
Fax:	(919) 531-9445
Email:	diane.hatcher@sas.com
Web:	www.sas.com