Implementing a Dimensional Data Warehouse with the SAS® System

Gregory S. Barnes Nelson
ASG, Inc.

Abstract

This paper will discuss the implementation of a dimensional data warehouse. A dimensional warehouse is a design or modeling technique that was developed by Ralph Kimball (Kimball, 1996) to help us develop our data models in a structured, visual way. Here we will discuss a strategy for the design, development and implementation of this structure using tools available with the SAS® System. In addition, we will talk about several modeling techniques, implementation of logical and physical schemas, transformation and aggregation strategies, and the loading and unloading of data to the warehouse.

Scope of this Paper

Data warehousing can mean a lot of things to a lot of people. This paper will provide a broad level perspective on the implementation of a simple data warehouse, or perhaps more accurately, a data mart. Occasionally, we will dip down to a technical level on specific issues and then resume our broad perspective.

The example we will be using throughout the paper will be data from the Northwinds Trading Company database that is distributed with Microsoft Access. The database is an order entry system for a company that sells a broad range of consumable/ food products. We selected this database for building our sample warehouse application for three reasons: (1) it is available to anyone who has MS-Access, (2) MS-Access is ODBC compliant and it was easy to get this data into SAS via SAS/ACCESS to ODBC, and (3) it provides a classic example of an operational system used in order processing that most of us have some familiarity with. After all, building a warehouse is typically done by extracting information from a transactional system and denormalizing the database (more on this later.)

We will first talk about the design phase of implementing our warehouse including discussions of what we mean by a dimensional warehouse. This discussion will carry us through some aspects of data modeling including the: business (process) model, logical (dimensional) and physical models we will create. This discussion will allow us to build our modeling skills and create the data structure for the start of a SAS application for the Northwinds Trading Company.

Modeling: Laying the ground work

Any architect tell you the most important part of building a literal, or physical warehouse is the plans. Before any mortar is poured or bricks are laid, the builders must have plans and specifications. As architects of our *data* warehouse, our job is to design a system that ends up with a tool that answers our original questions. Unlike our counterparts in the physical realm, the *data* architect builds a virtual world and has some flexibility with changes that will occur to the design. As, Ralph Kimball suggests, "The dimensional model is extremely robust. It can withstand serious changes to the content of the database without requiring existing applications to be rewritten" (Kimball, DBMS, August 1996).

We know that the design phase of a warehouse implementation is extremely important. Most of us are hard-core SAS programmers, teething on our early days of PROC Summary and the DATA step and haven't had any real training in building a data warehouse; much less formalized training as a data modeler. So how can we be expected to build a system that is flexible?

Types of Data Models

Data modeling is a science. Based on the use of well-defined terms and procedures, it is our job to build an understanding of our data. The data model helps us understand and even visualize relationships among our data.

During the planning and design phase of the data warehouse project, we really need to come up with a Requirements Definition Document. This document should describe the end-users expectations/needs, IT's expectations and needs, define what we want to try and tackle, a scope of work or work plan, and laying out some models to help us understand our problem (questions we want answered) and our data. This last piece is what I'd like to cover.

Models help us understand how things are connected, visually. Industry experts talk about a three model architecture (Inmon, 1993), and although they often refer to these models by different names, they all refer to similar concepts.

- Business Process Model
- Logical (Dimensional) model
- Physical model

Business Model

Goal:

- define the purpose, and decide on the subject(s) for the data warehouse
- identify questions of interest

The business model (Inmon calls this the high-level ERD or entity-relationship level), helps us understand what business questions we are asking. Here we focus on the what information is available; i.e., the attributes and relations between them. We do not try to put any thought into how the data should be accessed or organized, nor what it will be used for. For example, in Northwinds Trading Company database, we might want to try and answer some of these questions during the business modeling process:

1. Who bought the products (customers and their structure)?
2. Who sold the product (Sales organization, etc.)?
3. What was sold (product structure)?
4. When was it sold (time structure)?
5. What are the characteristics of the sale (discount, etc.)?

OLTP is from Mars vs. OLAP is from Venus

The way that we process information in a transactional system is very different from the way we want to analyze that data. In OLTP, or On-line transaction processing, systems such as financial, order entry, work scheduling, and point-of-sale, we want very fast response times with no redundancy and only the most current data on-line.

In contrast, data required by decision support analysts has a lengthy time horizon, is redundant to the support of varying data views, is often summarized and is non-updateable. In order to provide data to decision support analysts, relevant operational data is extracted from OLTP systems, cleansed, encoded (i.e., data and dimensions made to be consistent), and summarized. After being transformed into a format suitable for decision support, the data is uploaded into the data warehouse. The systems that we use to analyze data from the OLTP programs are called OLAP, or On-Line Analytical Processing. The data is organized very differently in these two scenarios.

OLTP uses a relational model in which all the data is broken out into separate tables to reduce redundancy, and eliminate duplicate key information, among other things. The process of "breaking" information out into separate tables is know as data *normalization.*

OLAP, on the other hand, needs the data efficiently structured for end-user reporting and decision support applications. As a result, the data tends to have a lot of redundancy. This process is called *de-normalization.*

Typically the way in which we talk about a transactional system (OLTP) is by reviewing its' ERD or entity-relationship diagram. Figure 1 illustrates the Northwind ERD. We can easily see the main subject areas and relationships. (In Appendix A, you can find the entire entity-relationship diagram for the Northwinds database).

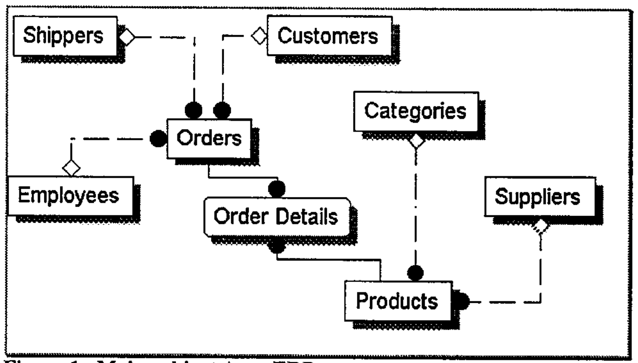


Figure 1. Main subject Area ERD

The diagram shows the relationships between tables in our database. Notice that the Order table is where most tables feed. This will be key in deciding which facts or subjects we want to model.

In our analysis tool (OLAP), we want the data to be relatively flat so we can calculate summary statistics without having to perform complex joins whenever we want a simple report. Table 1 shows an example of data that has been flattened out, or de-normalized. Note that the Order ID field contains duplicate records whereas in the relational model, this would be stored in a separate, normalized table.

Order ID	Order Date	Country	...	Salesperson	Extended Price
11044	20-Apr-95	Poland	...	Margaret Peacock	$591.60
11045	20-Apr-95	Canada	...	Michael Suyama	$37.50
...	...	...	...	...	...
11045	20-Apr-95	Canada	...	Michael Suyama	$1,272.00
11046	20-Apr-95	Germany	...	Laura Callahan	$722.00
11047	21-Apr-95	UK	...	Robert King	$480.37
11077	03-May-95	USA	...	Nancy Davolio	$26.00

Table 1. De-normalized data structure

Questions we can ask

Understanding our business processes and asking questions, will help us understand how our business is organized. This begins the development of our functional requirements. Figure 2 shows us the main subjects found in our operational data.

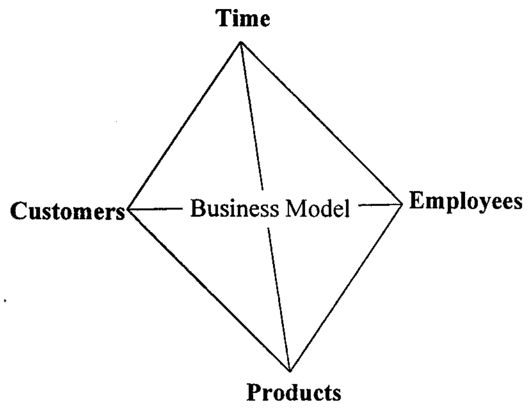


Figure 2. Main Subject Areas

As we will soon find out, the main subject areas are our "subjects" for analysis. At the intersection of each line, we can answer questions about our data.

Note that up to this point we have not talked about data we have in the system on our customers or our products. For example, we want to ask the following question:

What was the total revenue last year from people who were single versus those married?

In order to complete this we have to figure out what level of granularity (how far down does our detail data go) we have in our data.

When considering a measure such as revenue or sales, it is important to consider what data is available. Does sales information exist for each of my products? Does sales data exist for each of the sales reps? Does Sales data exist for the last five years? Metrics, such as Sales, do not exist in isolation, but rather in the context of dimensions such as Product, Employee, Customers, and Time which define what type of data is available. It is natural to think of data multi-dimensionally as shown in Figure 2. Sales revenue is the metric and is qualified by the dimensions of Customer, Product, Employee (Sales Rep) and Time.

Our data doesn't have marital status coded for each of our customers. The other issue is one of effective dating. An effective date in our operational system tells us something about the customer on a given date. For example, even if we had the customer's marital status, we would not have known their marital status for the time period we requested the data.

Dimensional (Logical) Model

During the analysis of the logical model we start to explore our functional requirements. What is it that we really want from the information we have about our organization, and how is it structured.

Goal:

- define our functional requirements
- confirm the subject areas
- figure out what the time dimension means
- identify the granularity (how deep can we go) for our subject(s)
- create "real" facts and dimensions from the subjects that we have identified

Our next step is to define and agree upon our subject areas that we identified in our business model. We developed some questions about our data. This led to categorizing these into our subject areas. Next we need to understand what time information we have and how far down we want to go with each subject. Here we decompose the data subjects into data entities comprising facts and dimensions (here we introduce the dimensional model).

A dimensional model is the proposed data modeling and design technique for the structuring of warehouse data. Ralph Kimball (Kimball, 1996) talks about the dimensional model (or start join schema) as if we could visualize it as a cube. The dimensions of the cube make up the dimensions of our warehouse. By using the image of a cube, the model can be better understood. "When a database can be visualized as a "cube" of three, four or even five or more dimensions," Kimball states, "people can imagine slicing and dicing that cube along each of its dimensions."

For example, in our data we know we have customer, products, employee and time. If we were to visualize this as a dimensional model, we would first visualize the three primary subjects: customers, products and employees in a cube.

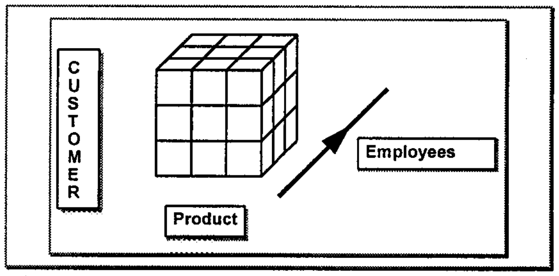


Figure 3. 3-D Model of our Main Subjects

Our three dimensional example can be extended to four dimensions by adding a time dimension to indicate the month of the year in which a sale was made. Visualizing a fourth dimension is more difficult than visualizing three. Imagine twelve boxes depicting the months of the year into which our cube is placed. When the box is placed in the JANUARY box, the cells contain information for JANUARY. When in the FEBRUARY box, the cells contain information for the month of FEBRUARY, and so on.

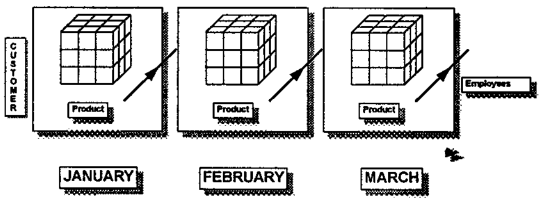


Figure 4. 4-D Model with the time dimension

This paradigm can be extended to five or more dimensions.

Kimball suggests that the data loaded into the warehouse is one of two types: a fact or a dimension. A fact is something that can be measured, they are numeric and generally continuous (e.g., sales) and are used to calculate some statistic. Dimensions, on the other hand, are pieces of information which categorize things. Examples here include, regions, sales people, fiscal quarter and so on.

Let's take this opportunity to review our example. In our data, the metric is sales revenue. We have four dimensions as depicted in Figure 4. At the intersection of two or more dimensions, we have a fact. For example, let's take the example of a two way intersection: *product B sold to customer 2.*

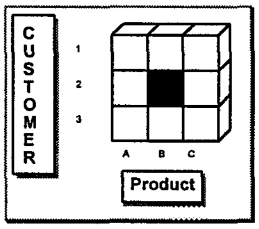


Figure 5. 3-D Model of our Main Subjects

If we calculate a summary statistic, such as sum of sales revenue, we could now answer: *how much money did customer 2 spend on product B?*

Obviously, we could come up with more complex questions for our warehouse which involves three or more dimensions. This is where the multi-dimensional database plays a significant role. The ability for users to slice-and-dice the data is done at a dimension level. *Ranging* is when users want to restrict their view to attributes on a dimension. In this scenario, a user might want just a few values from each dimension (e.g., I want to know how much revenue was generated by two of my employees from my two largest customers in the beverage and produce categories.) Figure 6 shows this situation.

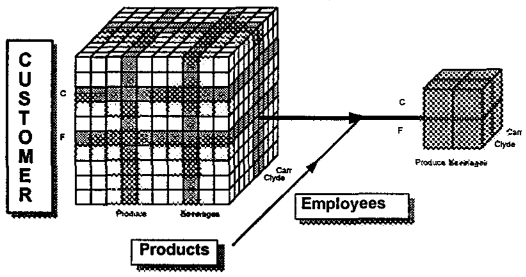


Figure 6. 3-D Model of our Main Subjects

In sum, the logical model helps us think about how our data is organized and whether or not we have the granularity in our data to answer the question(s). The next phase is to develop the physical model.

Physical model

The physical model in the warehouse design gets into the nitty-gritty of what data we have available and how should it be stored. The physical design of our data is called a schema. A schema for our data warehouse can be represented by one or more design constructs such as:

- entity-relationship model
- star schemas
- snowflake schemas
- fact-constellation schemas
- persistent multidimensional stores
- or summary tables

Others have talked about the use of such modeling techniques (e.g., Betancourt, 1996; Kimball, 1996; McGuff, 199 and Red Brick, 1995; Raden, 1995), so I will skip this and let you review those articles for your own edification.

Goal:

- define the actual storage architecture
- decide on how the data is to be accessed and how it is arranged

In considering the data warehouse data model, it is useful to first review the types of tables found in a data warehouse. The three types of tables are:

- Primary Data Tables
- Descriptor Tables
- Characteristics Tables

Primary Data Tables contain both metrics and attributes, and contain the data that end-users are seeking. In Figure 7, the Primary Data Table contains the attributes: ProductID, CustomerID, EmployeeID and the metric Sales. In large data warehouses, the full-text attribute description is not stored in the Primary Data Table, but rather in a Descriptor Table. Numeric element ID codes are stored in the Primary Data Table. These numeric ID codes index faster, yield smaller indices, and provide faster "WHERE" clause matching.

Descriptor Tables often contain only two columns, the attribute ID code and the common-English description of the attribute. There is a one-to-one relationship between the ID and the description. These tables replace the ID codes used in queries with common business terms familiar to the user. In Figure 7, there are three Descriptor Tables which map ProductID, CustomerID and EmployeeID codes to their respective user-understandable business terms. In smaller warehouses, where load performance and storage concerns are less of a problem, text descriptors may appear in the Primary Data Tables this increases data comprehensibility.

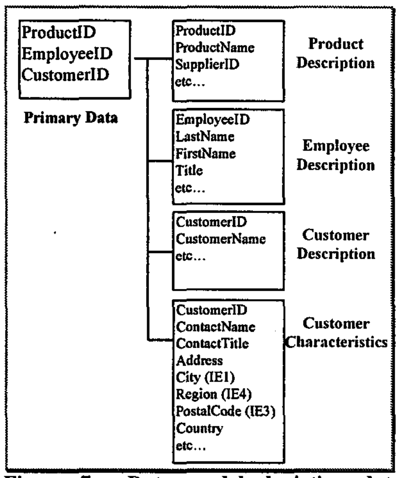


Figure 7. Data model depicting data warehouse tables

Characteristics Tables contain additional information about an attribute and can be used to segment data in an ad-hoc manner. Each column in a Characteristics Table represents an additional attribute that can be used as a filtering criterion in queries. In Figure 7, there is a single Characteristics Table which contains additional attributes related to the Customer attribute. Using this Characteristics Table, Sales could be segmented by the location of the customer, or any other attribute in the Characteristics Table.

Our task is now to develop the databases to help us answer our questions. The schema we use will guide our programming tasks to that end. The star-schema and the snowflake are the most widely use schemas.

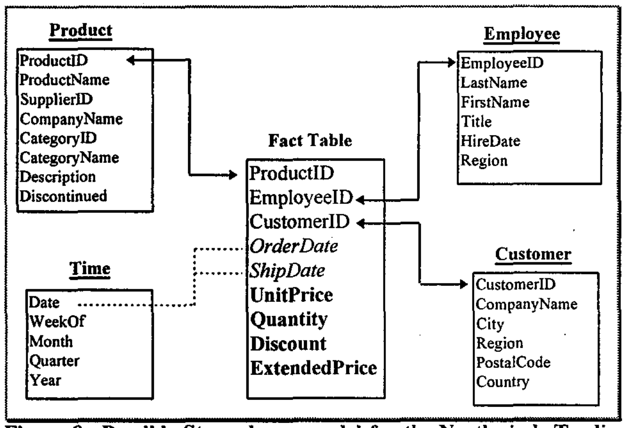


Figure 8. Possible Star-schema model for the Northwinds Trading Company system.

The star schema model in figure 8 is based on a central Fact Table surrounded by any number of Dimension Tables. The fact table has few columns or variables but many observations. These are linked by dimension tables that are typically short and wide, that is, many columns, few observations.

The snowflake schema is similar to the star schema but normalizes our data by breaking out the lookup tables (called *outrigger* tables). One example of the data in the snowflake schema is shown in Figure 9.

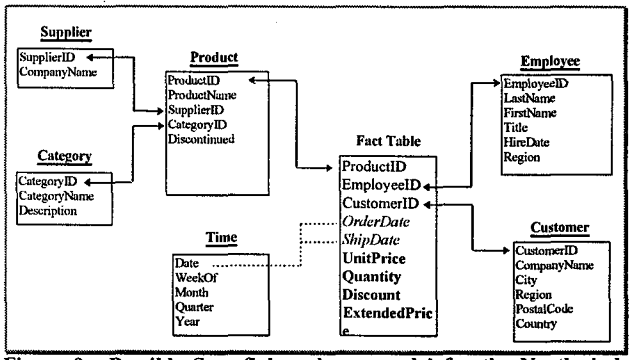


Figure 9. Possible Snowflake schema model for the Northwinds Trading Company database.

Here the snowflake schema is normalized even more than the star schema by breaking out the categories and supplier information into outrigger tables. The normalization of the dimension tables reduces storage overhead by eliminating redundant data values in the dimension table. Normalization comes at the cost of complex queries which are often more difficult to use and implement and require more processing at run-time.

Implementing the Warehouse

There are a variety of tasks that are required in order to build a successful warehouse. We've reviewed the steps in building our models which helped us understand and narrow our questions. The logical and physical models helped us structure the data for the end-user application.

Now we're ready to start building our physical model. The tasks now at hand are:

- Extraction
- Data Validation
- Scrubbing (cleansing)
- Integration
- Structuring
- Denormalizing
- Summarizing
- Create fact and dimension tables
- Optimize our indexes and queries
- Create views
- and finally, develop an exploitation methodology that takes advantage of the technology of multidimensional databases, EIS, and other query tools.

There is no way that we could discuss all of these topics in this paper. Instead, we'll focus on what the users will see and how we can structure the data in ways that are efficient for both the end-user and the computer.

Extraction

Complicated data warehouse scenarios might entail pulling data from multiple systems (e.g., VSAM, Oracle, Excel, SAS datasets, etc.) and bring them together in a common fashion. In the Northwinds example, we only have one data source that we pull data from, albeit multiple tables. We are going to build a fact table and join some dimension information by using the SQL pass-through facility in SAS. I will also illustrate pulling a lookup table from Microsoft Access and storing it as a SAS data view rather than a data set.

Creating our Fact Table using the SQL Pass-Through Facility

We need to pull several columns from the relational database to form our fact table. After review, let's say we decide to use the star-schema for our physical design (Note: prior to running this code, be sure to use the ODBC Administrator to set up the Microsoft ODBC engine to Access). I named my ODBC database: Northwind.)

```
Proc SQL;
 connect to ODBC (DSN=Northwind);
 create table work.fact as
 select *
 from connection to ODBC

  (
  SELECT DISTINCTROW
    Products.ProductID,
    Employees.EmployeeID,
    Customers.CustomerID,
    Orders.OrderDate,
    Orders.ShippedDate,
    [Order Details].UnitPrice,
    [Order Details].Quantity,
    [Order Details].Discount,
    CCur([Order Details].[UnitPrice]*
    [Quantity]*(1-[Discount])/100)*100 AS ExtendedPrice
   FROM
    Shippers INNER JOIN
     (Products INNER JOIN
      ((Employees INNER JOIN
        (Customers INNER JOIN Orders
          ON Customers.CustomerID = Orders.CustomerID)
      ON Employees.EmployeeID = Orders.EmployeeID)
          INNER JOIN [Order Details]
          ON Orders.OrderID = [Order Details].OrderID)
     ON Products.ProductID = [Order Details].ProductID)
    ON Shippers.ShipperID = Orders.ShipVia
  );

 disconnect from ODBC;
 quit;
```

Here I show the SQL pass-through method that creates a table called WORK.FACT. This will generate 2155 rows and 9 columns. Again, a fairly long and narrow table. You can use a variety of methods to pull the data to form a fact table.

Creating our Dimension Tables using SQL Views

The next step is to extract the dimension tables. Here we create our three "real" dimensions (product, employee and customer) and perform the extraction's via SQL data views. There are a variety of methods we can use, but I like SQL and am familiar with Microsoft Access' SQL extensions. Note that in the SQL pass-through facility, we use the proprietary or native SQL of Microsoft Access, not SAS.

```
Proc SQL;
 connect to ODBC (DSN=Northwind);
```

```
 /* Products */
 create view work.product as
 select *
 from connection to ODBC
  (
SELECT DISTINCTROW
        Products.ProductID,
        Products.ProductName,
        Products.CategoryID,
        Categories.Description

  FROM Categories INNER JOIN Products
       ON Categories.CategoryID = Products.CategoryID
  );

/* Customer */
 create view work.customer as
 select *
 from connection to ODBC
  (
SELECT  [CustomerID],
        [CompanyName],
        [City],
        [Region],
        [PostalCode],
        [Country]
FROM Customers
  );
/* Employee */
 create view work.employee as
 select
        *,
        datepart(HireDate) as hdate format=date7.
 from connection to ODBC
  (
SELECT  [EmployeeID],
        [LastName],
        [FirstName],
        [Title],
        [HireDate],
        [Region],
        [Country]
FROM Employees
  );
 disconnect from ODBC;
 quit;
```

Time Dimension

We really haven't talked out the time dimension yet. Our time dimension is a sort of virtual dimension since we have a date stamp in which the order was placed (OrderDate) and a date stamp when the order was shipped (ShipDate). In order to effectively report on a given time period such as calendar week, month, fiscal year and so on, we can create the data needed for our time dimension with the following code.

```
PROC SQL;
 reset noprint;
 select min(min_ord, min_ship),
      max(max_ord, max_ship) into :min_date, :max_date
    from (
         select
                 min(datepart(orderdat)) as min_ord,
                 max(datepart(orderdat)) as max_ord,
                 min(datepart(shippedd)) as min_ship,
                 max(datepart(shippedd)) as max_ship
              from work.fact
              )
 ;
QUIT;

Data dates (drop=_i rename=(date=start weekof=label) );
  fmtname='calweek';
  do _i =&min_date to &max_date by 1;
        date=_i;
        weekof=intck('week',intnx('year',_i,0),_i)+1;
        output;
  end;
run;

Proc format cntlin=dates;
run;
```

Note that the code here uses the order and ship dates to determine the date range and creates two macro variables (&min_date and &max_date) which contain the highest and lowest date values in our data. We use this as input for creating a format called calweek, that we will use later.

Data Transformations

Once we have all of our data extracted into our physical model (e.g., star-schema), we are ready to start structuring the data for our analysis application. Up to this point, we only have the lowest level of detail, or granularity based on our fact table. The level of granularity in our fact table is a single product ordered by a customers, sold by a sales rep. Alternatively, we could have rolled up our fact table to the order which would have given us all products ordered by a given customer at a particular time. The way in which we structure the data is dictated by what kind of answers we look for. In this case, what would be the implications of this structure?

For performance reasons, we have to decide how we want to summarize our data. Normally at this step, we would do our validation checks to look for invalid or missing data, out-of-range values, duplicate records on our keys, etc.

Let's talk for a moment about integration of our data. As I said before, our data is coming from a single source: the Northwinds Trading Company order entry application. In the "real world" we would probably integrate this data with other information. For example, if we had additional information about employees in the HR system or customer information in the accounting system we would most likely want to integrate that information into our warehouse.

Integration can be summed up pretty simply: in order for the data to be of any use, there must be consistency in the way we code the dimensions. Remember, data values must be consistent (e.g., region numerically coded in the Sales revenue system, but character codes are used in the accounting system). Also, in the fact table, the metrics must be scaled similarly in order to make any sense of the values they represent. Typically in the SAS world, we can achieve consistency in one of the following ways:

- The SQL procedure
- DATA step statements
    - FORMAT
    - INFORMAT
    - LABEL
    - RENAME
- DATA step options
    - RENAME

The last piece in our transformation strategy would be de-normalize our data. In the snowflake or star schema we have done some de-normalization, but now we have to make it even flatter. Combine the fact and dimension tables to form a subject view of the world. Again in SAS, we typically do this by joining tables with the SQL procedure or by merging data sets with the MERGE statement. For example, in our data, if we want to pull in a product description instead or in addition to the product ID, we could use the following code to merge in the product and category descriptions.

```
Proc sql;
  create table proddesc as
        select
        a.*,
        b.ProductI,
        b.ProductN,
        b.Category,
        b.Descript

FROM    fact    as a,
        product as b
WHERE   A.PRODUCTI=B.producti
  ;
QUIT;
```

Once we've done this for all of our dimension tables, we should have a single table that contains all of the information we need and we're ready to start summarizing or aggregating our data.

```
Proc sql;
  create table final as
        select

        a.*,
        b.ProductN,
        b.Category,
        b.Descript,
        c.firstnam||' '||c.lastname as salesrep,
        c.title,
        c.region,
        c.country,
        c.hdate as hiredate,
        d.companyn,
        d.city as custcity,
        d.region as c_region,
        d.postalco as c_zip,
        d.country as c_cntry,
        datepart(a.orderdat) as weekof format=calweek.,
        datepart(a.orderdat) as qtr format=qtr.,
        datepart(a.orderdat) as year format=year.


FROM    fact    as a,
        product as b,
        employee as c,
        customer as d

WHERE   (A.PRODUCTI=B.producti) and
        (a.employee=c.employee) and
        (a.customer=d.customer)
  ;
QUIT;
```

This code (see above) de-normalized all of the tables into one central table, ready for summarization in a single SQL step. Note that we have representation from all four dimensions in this de-normalized table including the calendar week and year, even though we didn't have these in any of our original tables.

Aggregation Strategies

Once we have a fully de-normalized table, we can now start to think about how we want to roll-up or summarize our data for analysis. We can have the largest impact on performance by utilizing optimization techniques that take advantage of pre-summarizing our data. Knowing what to summarize and what to leave to the user at run-time, will be something that you hopefully gleaned from your interviews with the users. For example, what dimensions are they most interested in seeing (e.g., total sales by customer) or what crossings are most important to them (e.g., total sales for each customer by product category.) Let's proceed as if we had some idea about how the end-users want to slice and dice the data.

In SAS, there are at least three methods of summarizing data for our purposes:

- SQL and the GROUP-BY option
- PROC SUMMARY
- PROC MDDB

I will discuss each of these by example.

A word about Hierarchies, drill-downs and dimensions

In our warehouse, we know we have sales information for our four dimensions: Time, Customers, Employees and Products. Each of these have several levels that we know our users will be interested in seeing. Table 2 shows us the four dimensions and their hierarchies.

Employee	Time	Customer	Product
Employee ID	Order Date	Company Name	Product ID
Region	Calendar Week	Region	Category
Country	Quarter	Country	
	Year		

Table 2. Northwinds warehouse dimensions

For example, take note of Figure 10 below to review the employee hierarchy.

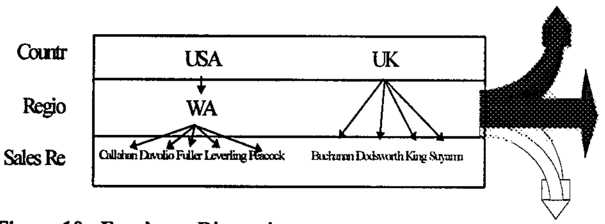

Countr
Regio
Sales Re
USA
UK
WA
Callahan Davolio Fuller Leverling Peacock
Buchanan Dodsworth King Suyama

Figure 10. Employee Dimension

Notice that region only has meaning in the USA, so we need to consider that when adding a region level to the data. To review the hierarchies in SAS, you can simply perform crossings with PROC FREQ or PROC SUMMARY. For example, the code below helps us understand the EMPLOYEE hierarchy.

```
 Proc summary data=final;
  class salesrep region country;
  var extended;
  output out=sumstat sum=extended;
run;
```

This produces a summary output dataset containing all of the possible combinations of salesrep, region and country. We can use this in our end-user application for very quick manipulations among the different levels within the dimensions. Navigating the employee hierarchy is referred to as "rolling-up" and "drilling-down".

In Figure 10, we saw that users can roll-up and drill-down through a single dimension, EMPLOYEE. Well designed warehouses also allow users to roll-up and drill down through multiple dimensions concurrently. Thus, in this example, an end-user could hold the products, customers and employees constant, while drilling-down or rolling up through sales figures over the TIME dimension.

In addition to PROC SUMMARY as a method for creating these aggregate or summary tables, we can also use PROC SQL or PROC MDDB. The following codes shows these methods.

```
Proc sql;
  create table summary as
        select salesrep,
               region,
               country,
               sum(extended) as extended,
               count(*) as freq
  from work.final
  group by salesrep, region, country;
quit;
```

We used PROC SQL to create a summary table for salesrep by region by country. Although, not as comprehensive as PROC SUMMARY, PROC SQL has a tremendous amount of power when it comes to the calculation of new information when you create these roll-ups.

Finally, a new entry into the SAS family of products, PROC MDDB provides methods to build a multi-dimensional database with multiple hierarchies. The following code shows an example of how we could build the employee, product, customer and time hierarchies with PROC MDDB. For simplicity, I included only the time and customer hierarchy here.

```
Proc mddb data=work.final out=warehous.mymddb;
     class year qtr weekof country region;
     var extended / sum min max;
     hierarchy year country;
     hierarchy year qtr country;
     hierarchy year qtr weekof country;
     hierarchy year country region;
     hierarchy year qtr country region;
 run;
```

The code was built to optimize drill-downs in our application. To achieve an optimal set of subtables, we crossed the drilldown hierarchies so that no matter where the user is in the hierarchy, there is a subtable with the appropriate summary information. Table 3 shows us this scenario.

Time Hierarchy			Customer Hierarchy	
Year	Qtr	Weekof	Country	Region
x			x	
x			x	x
x	x		x	
x	x		x	x
x	x	x	x	
x	x	x	x	x

Table 3. Sample Hierarchy Structure for Northwinds MDDB.

Note that when we build a MDDB dataset, we can only access data through procedures specifically designed to access this data (e.g., 3D Graph, Multidimensional Report, Org Chart and the Graphic Variance EIS objects.)

Loading and Refreshing

After you have extracted and transformed your data into your physical structure, de-normalized and created the summarization scheme, you are ready to load the data into the warehouse. There are a variety of methods to take data from your created structures and develop tools for the users to analyze that information. Betancourt (Betancourt, 1996) tells us that the designer must consider three different loading strategies:

1. The loading of data that is already archived or off-line;
2. the loading of data contained in existing applications; and
3. incremental changes from the operational system since the last time the data was loaded.

We wont go into detail about these strategies in this paper. However, one of the most exciting advancements in SAS is a method introduced with PROC MDDB to "drip-feed" or update our MDDB dataset. For this, we use the MDDB= option on our PROC MDDB statement.

For example, to update our WAREHOUS.MYMDDB dataset, we used the following syntax:

```
proc mddb data=work.new out=warehous.newmddb
mddb=warehous.mymddb;
```

This would create a new MDDB data set from the new detail information.

Conclusion

The goal of this paper was to provide the reader with a practical guide to data modeling techniques and their implementation with SAS software. Data warehousing is a complex topic; my hope is that you leave with a better understanding of the issues surrounding the design of a warehouse and what implications those early design decisions have on the decision support analysts.

We discussed the dimensional model that that helped us think about, and organize our thoughts about, data warehousing. The dimensional data warehouse provides not only a framework for us, but is also a practical guide which fits well with tools in SAS software.

As the SAS toolset becomes more mature, we hope to see additional tools to help us manage not only the physical implementation of our schema(s), but also the business and logical modeling components.

Acknowledgements

We would like to acknowledge the assistance of ASG, Inc., my employer, with providing me the time and resources for developing this paper. Also, a special thanks to John Anderson, Randy Betancourt, and Mark Mooreman for their assistance in reviewing this manuscript.

Bibliography

Betancourt, R. "The SAS System in a Data Warehouse Environment". Published in the proceedings of the annual convention of the South Eastern SAS Uusers Group. 1996

Inmon, W.H. "Building the Data Warehouse". John Wiley and Sons 1993. ISBN 0-471-56960-7

Inmon, W.H. "Information Systems Architecture" QED.

Kimball, R. "The Data Warehouse Toolkit: Practical Techniques for Building Dimensional Data Warehouses". John Wiley and Sons 1996. ISBN 0-471-15337-0

Kimball, R. "Dangerous Preconceptions: Discovering the liberating truths that can lead to a successful data warehouse project". DBMS Magazine. August 1996 from http://www.dbmsmag.com/9608d05.html.

McGuff, F. "Data Modeling for Data Warehouses" October, 1996 from http://members.aol.com/fmcguff/dwmodel/dwmodel.htm

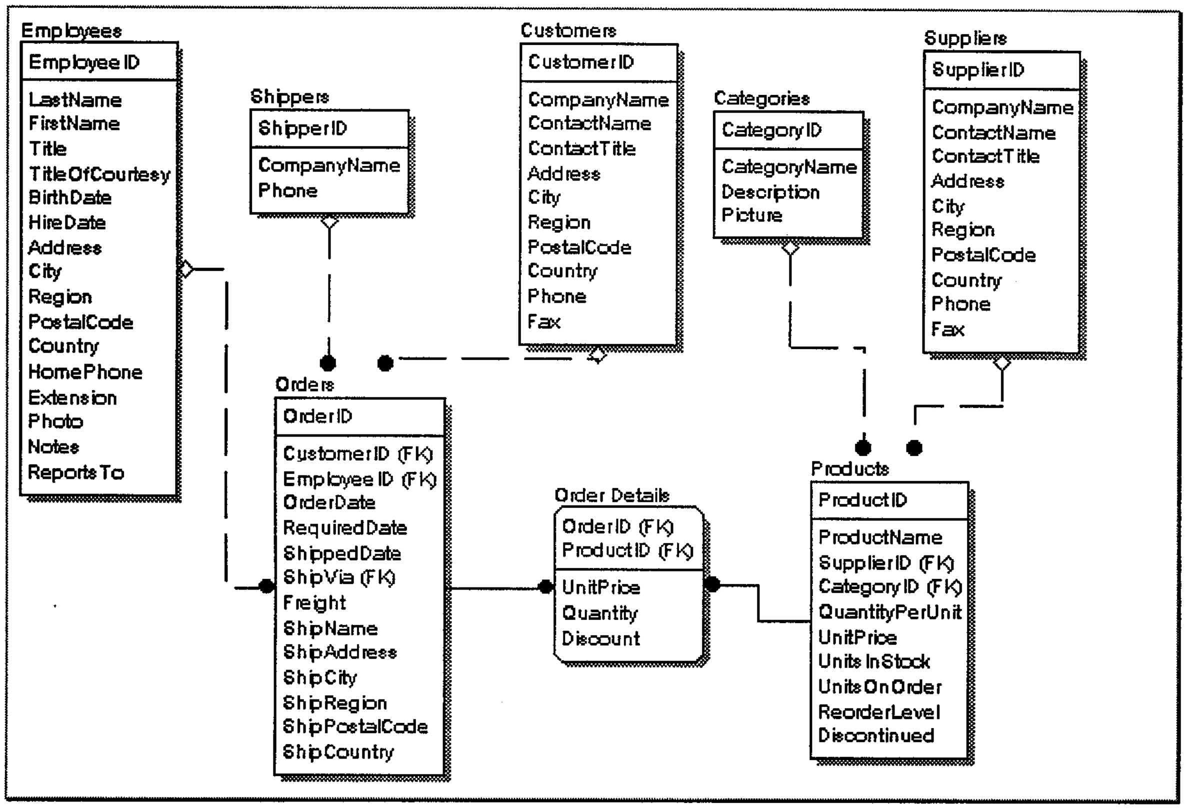
Red Brick "Star Schemas and STARjoin Technology" White Paper from http://www.logicworks.com. 1995

Address

Questions pertaining to this article should be addressed to:

Gregory S. Barnes Nelson
2000 Regency Parkway, Suite 355
Cary, NC 27511
919-467-0505
919-467-2469 (Fax)
gbn@asg-inc.com

Appendix A. Entity-Relationship Diagram for the Northwinds Trading Company Database.


Employees
EmployeeID
LastName
FirstName
Title
TitleOfCourtesy
BirthDate
HireDate
Address
City
Region
PostalCode
Country
HomePhone
Extension
Photo
Notes
ReportsTo
Shippers
ShipperID
CompanyName
Phone
Customers
CustomerID
CompanyName
ContactName
ContactTitle
Address
City
Region
PostalCode
Country
Phone
Fax
Categories
CategoryID
CategoryName
Description
Picture
Suppliers
SupplierID
CompanyName
ContactName
ContactTitle
Address
City
Region
PostalCode
Country
Phone
Fax
Orders
OrderID
CustomerID (FK)
EmployeeID (FK)
OrderDate
RequiredDate
ShippedDate
ShipVia (FK)
Freight
ShipName
ShipAddress
ShipCity
ShipRegion
ShipPostalCode
ShipCountry
Order Details
OrderID (FK)
ProductID (FK)
UnitPrice
Quantity
Discount
Products
ProductID
ProductName
SupplierID (FK)
CategoryID (FK)
QuantityPerUnit
UnitPrice
UnitsInStock
UnitsOnOrder
ReorderLevel
Discontinued