

SAS Table Design – The Good, The Bad, And The Ugly

Kenneth Bissett

United Services Automobile Association (USAA)

Abstract

How do you go about designing your SAS tables? What defines a row and what defines a column? Should you have a row for each unique customer account, or should you store one row per customer, with a column for each of their accounts? Does it even matter?

In the database world, the process of table design is called database normalization. A highly skilled database administrator will design a series of tables to reduce data redundancy, lower storage requirements, and hopefully speed up processing. In the world of SAS, we typically don't consider the highly skilled DBA because we're usually not building complex sets of related tables. But there are things we should consider.

This paper will give the SAS programmer a basic idea of some of the table design considerations as well as provide some business scenarios where good table design is critical. Examples will also be provided which will show different techniques for analyzing data depending on the table design.

A Scenario

Designing your SAS tables properly may never have entered your mind, but it's very important to create a good design up front in order to avoid problems. Here's a common scenario: You need to store data on a set of customers and which products they own. If your company has 5 different products, you could create a column for each product. Now you can easily sum up total products owned by each customer using a sum function. But what if your company has hundreds of products – and those products are constantly changing? Now you would need to create hundreds of columns for those products. If the average number of products for each customer were 3.5, then very few of those hundreds of product columns would have any data in them. This is a serious waste of disk space, not to mention the maintenance of keeping track of the column names. You might also need to store a date associated with each product, which means you now need two columns for each product! There is an alternative.

Where do I start?

Getting started with good table design can be a challenge if you are unaware of the pitfalls of a bad design. If you know what *not* to do, it can sometimes lead the way to a better design with little effort. On the other hand, it is not always a bad idea to have a less than perfect table design depending on the types of reporting and analysis required.

A good place to start is to consider what kinds of reports and analysis will be required. For example, a SAS dataset will be used to store information about customers and the products they own. A normalized version of that SAS table might have a customer id and a product name. There will be one record per customer, per product.

Table1

CustomerID	ProductName
1	AccordEX
1	Wrangler
2	CelicaGT
2	F150
3	F150
4	CivicLX
4	Wrangler
5	CivicEX

This table could also be stored in a denormalized structure:

Table2

CustomerID	AccordEX	Wrangler	Celica GT	F150	CivicLX	CivicEX
1	1	1	0	0	0	0
2	0	0	1	1	0	0
3	0	0	0	1	0	0
4	0	1	0	0	1	0
5	0	0	0	0	0	1

Notice the difference in the layout of the table. A column for each unique product is required. However, all of the data for each unique customer is on a single record. Let's take a look at a simple report using each source.

```
TITLE LONG AND SKINNY;  
PROC FREQ DATA=TABLE1;  
TABLES PRODUCT;  
RUN;
```

LONG AND SKINNY

The FREQ Procedure

PRODUCT	Frequency	Percent	Cumulative Frequency	Cumulative Percent
ACCORDEX	1	12.50	1	12.50
CELICAGT	1	12.50	2	25.00
CIVICEX	1	12.50	3	37.50
CIVICLX	1	12.50	4	50.00
F150	2	25.00	6	75.00
WRANGLER	2	25.00	8	100.00

```
TITLE SHORT AND WIDE;  
PROC MEANS DATA=TABLE2 SUM MAXDEC=0;  
VAR ACCORDEX--CIVICEX;  
RUN;
```

SHORT AND WIDE

The MEANS Procedure

Variable	Sum
ACCORDEX	1
WRANGLER	2
CELICAGT	1
F150	2
CIVICLX	1
CIVICEX	1

Both reports create the same basic information from two entirely different table layouts. Another report might be the average number of products per customer.

```
TITLE LONG AND SKINNY;  
PROC FREQ NOPRINT DATA=TABLE1;  
TABLES CUSTOMERID / OUT=TEMP;  
RUN;  
PROC MEANS DATA=TEMP MEAN MAXDEC=1;  
VAR COUNT;  
RUN;
```

LONG AND SKINNY

The MEANS Procedure

Analysis Variable : COUNT Frequency Count

Mean
1.6

```

TITLE SHORT AND WIDE;
PROC SQL;
SELECT
AVG(ACCORDEX+WRANGLER+CELICAGT+F150+CIVICLX+CIVICEX)
  AS TOTCARS
FROM TABLE2;
QUIT;

```

Mean
1.6

Both of these reports produce the same result. Notice in the second example using PROC SQL, a sum of all product variables is needed. Imagine this scenario with hundreds of product variables.

Table Maintenance

Using the two table designs above, what kind of maintenance challenges exist for Table1 and Table2? Most companies don't keep the same set of products forever. New products can be added, old ones can be removed, and existing products can be changed. In Table1, the record contains the name of the product as a text value in the field called ProductName. If the company adds a new product, the text is added as a new value in the ProductName column. No record exists for the customer until they purchase the new product. On the other hand, in Table2 there is no column for the new product. A new column will need to be added to the existing dataset (by default, any preexisting records will have a missing value in the new product column).

In a true relational database design, there would likely be a small lookup table that stores each of the products and some of their attributes. This table would be updated as products change and would be linked to a customer product table using a product key for example. In the SAS world, we can use a format as a product "lookup table" and update that format as products change.

The maintenance of Table2 would be more challenging. The main reason being that a new column is required whenever a new product is added. This also produces some challenges in an automated environment. If a series of programs were executed against Table2, some level of manual modification is necessary to make the programs aware of the new column.

A More Advanced Scenario

Using the example from above, we can use product codes instead of product names. Table1 would look the same with the exception of codes in the product column rather than product names.

Table1

CustomerID	ProductName
1	101
1	102
2	103
2	104
3	104
4	105
4	102
5	106

With the above table design, we are using product codes that would likely need to be formatted to be of use in a report. We can assign the product names to the codes in a format library and maintain the formats for new or changed product codes. We can also assign different formats such as product categories to those same codes:

```
PROC FORMAT LIBRARY=WORK;
  VALUE PRODFMT      101 = 'Honda Accord EX'
                    102 = 'Jeep Wrangler'
                    103 = 'Toyota Celica GT'
                    104 = 'Ford F-150'
                    105 = 'Honda Civic LX'
                    106 = 'Honda Civic EX';

  VALUE PRODGRP      101, 105, 106   = 'Honda'
                    102              = 'Jeep'
                    103              = 'Toyota'
                    104              = 'Ford';

RUN;
```

With these formats, it is now possible to see the products distributed or counted in more than one way. This allows the data to be consistently stored while providing flexibility in your analysis approach.

For Table2 it might be easier to manage Prod1 through Prod6 as column names rather than using the actual name of the product. Of course this also might make it more challenging for an end user to remember which column goes with which product.

Table2

CustomerID	Prod1	Prod2	Prod3	Prod4	Prod5	Prod6
1	1	1	0	0	0	0
2	0	0	1	1	0	0
3	0	0	0	1	0	0
4	0	1	0	0	1	0
5	0	0	0	0	0	1

The SAS language has features that make it easier to list all of the column names with numeric suffixes:

```
PROC FREQ DATA=TABLE2;
TABLES PROD1-PROD6;
RUN;
```

In this scenario, we would need to label the columns appropriately with the actual product name:

```
LABEL      PROD1 = 'Accord EX'
           PROD2 = 'Wrangler'
           PROD3 = 'Celica GT'
           PROD4 = 'F-150'
           PROD5 = 'Civic LX'
           PROD6 = 'Civic EX';
```

Instead of grouping similar products using a SAS format, in this case we will need to group similar products by column name:

```
HondaTot =      PROD1 + PROD5 + PROD6;
JeepTot =       PROD2;
ToyotaTot =     PROD3;
FordTot =       PROD4;
```

This is not so bad in this small example, but imagine this design with 50, or even 100 different products and multiple product groupings. SAS arrays could also be used to group similar product columns but the challenge of column management still remains.

Conclusion

SAS programmers are in a unique position in many ways. One of those is the ability to manage their own data. Along with managing your own data come the challenges of organizing that data so that it is not only stored efficiently and is easy to maintain, but also to allow easier access to the data by the end users. The trade-off between these two disciplines should be understood before any decisions are made. Of course, with SAS, you can change your mind and re-organize your data quite easily – or even create more than one version of the data to allow different access patterns from different users.

Don't forget about the possibility of working closely with a database administrator who may be able to help you organize your SAS tables. A DBA can help you understand how to break large records into separate tables and smaller records that could make your data easier to access and maintain.

Contact

Kenny Bissett
United Services Automobile Association (USAA)
Kenneth.bissett@usaa.com