Paper 47-26

One Engine, Three Apps – Rapid Application Development By Not Writing Code

Amy Swinford, Trilogy Consulting, Lehi, UT

ABSTRACT

The original business need was to develop a customized query tool for a data warehouse that did not yet exist. The data was expected to be large – millions of observations with thousands of variables. Another department was creating the database and their design for efficient data storage made getting information cumbersome for the uninitiated. The solution was a data-driven, custom query tool that did not require a degree in database engineering to use. Users were able to answer questions that previously took weeks and a programmer to resolve, as well as some that couldn't even be approached. The added benefit turned out to be an interface that could be easily configured for other datamarts. The interface works on any Windows based system, and has been successfully tested on Unix systems as well. The data it grabs has, at various times, been located on Windows, Unix, and MVS servers. The same engine has now been in use for 3 different applications and a fourth is on the way.

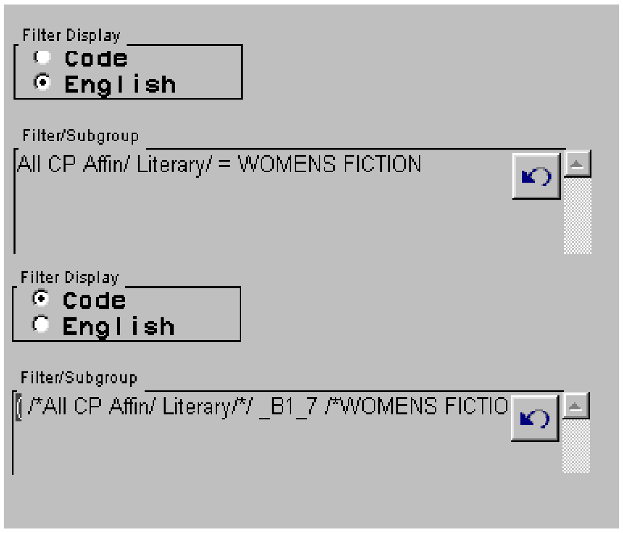
THE PROBLEM

With multiple lines of business operating independently, the client was facing the reality that they needed more intelligence in their marketing promotions. When marketers needed to choose a population for a new promotion, they would put in a request for a count of prospects that would be sent to a programmer. Days, and even weeks later, they would get the numbers on which to base their decision. If the results raised new questions, the new request would go through the same arduous process. Answers were not being provided in a timely manner, as well as it being expensive to provide programmers to write the query code for each request. Data was stored in disparate locations, and decisions were based on scant information, especially when considering the rich data available. Something was obviously required to make promotions more effective with less money. The answer to this was to create a new data warehouse that would combine data from many lines of business, providing more intelligence to each decision process. Moreover, the information would be almost immediately available; allowing for better decisions based on current conditions.

Then came the issue of how to actually get to the data. Before the client had asked for assistance, the decision had already been made to create the warehouse in Oracle. Initial testing with various query tools had proved unsatisfactory. Users needed a deep understanding of databases and query construction in order to get to the data themselves. Once again a situation was being constructed that required programmer intervention to answer simple questions. Enter SAS. By creating a datamart in SAS and combining it with a custom query tool, we hoped to answer the majority of these questions without a programmer. The wrinkle to this was that we needed the query tool the instant that the datamart became available. The warehouse was estimated to be available in 2 months time, none of the query tools available for SAS were suitable to our client, and we needed to work fast.

REQUIREMENTS

The client originally wanted three basic capabilities with the query tool. First, and most difficult, is the ability to select the data of interest using a simple to use interface. Simple interfaces are, without a doubt, the hardest to write. For a marketer, even the concept of a variable can be foreign. Add to this the need to be able to do complex queries. Some analysts were sophisticated enough to require complex queries built of subqueries. So an interface needed to be point-and-click, drag-and-drop, but still allow for extremely complex queries. Most interfaces on the market are either one or the other: simple, or meant for sophisticated users, not both. To meet this requirement, the tool provides users the ability to create a query using a simple drop and drag technique, along with buttons that appear as needed to guide the user through defining a query. A view is also provided to switch back and forth between an "English" query and a "Code" query. Only the code version allows users to type directly into the query.

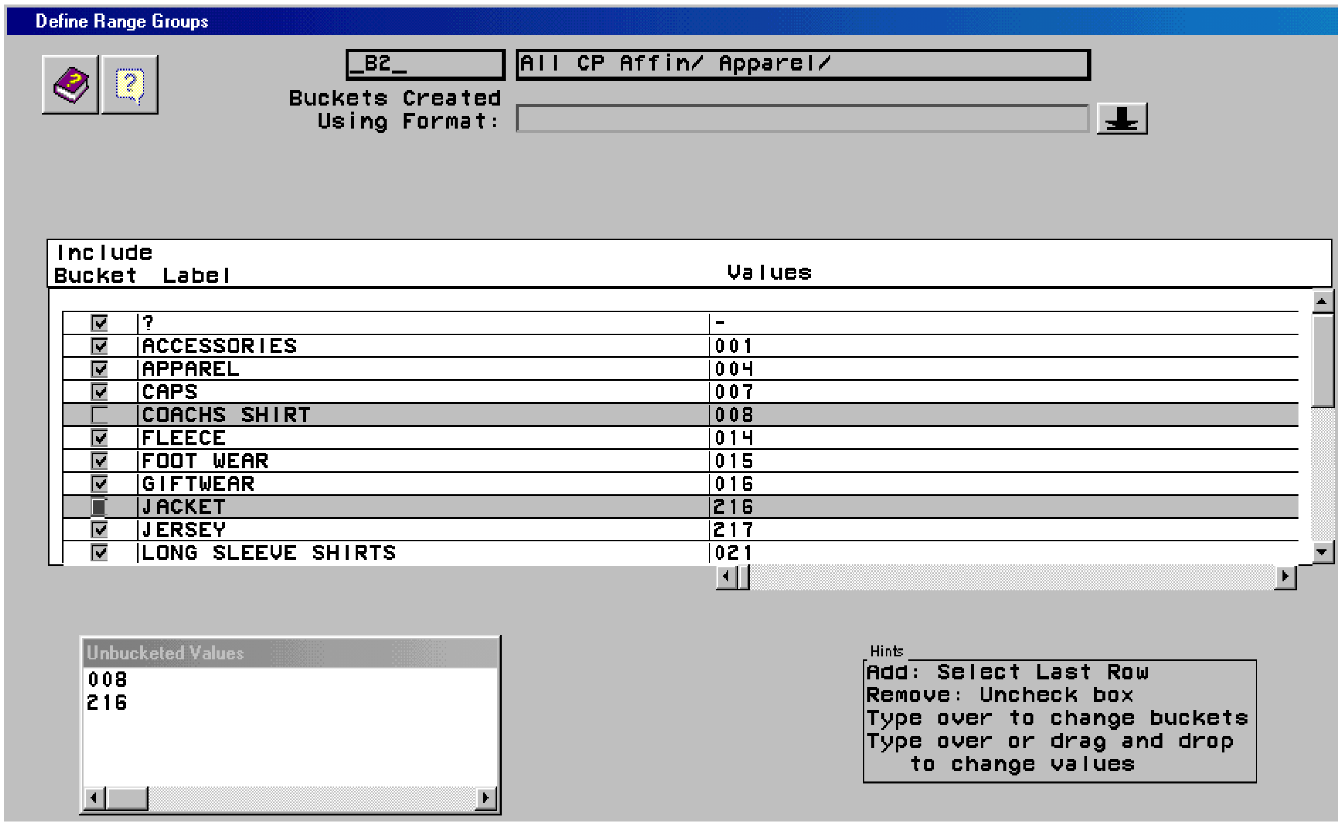

Filter Display
( ) Code
(•) English

Filter/Subgroup
All CP Affin/ Literary/ = WOMENS FICTION

Filter Display
(•) Code
( ) English

Filter/Subgroup
[/*All CP Affin/ Literary/*/ _B1_7 /*WOMENS FICTIO

The second necessary capability was the ability to define groups. These groups could be based on character variables, continuous variables, or a combination thereof. And they needed to be ad-hoc, so if we want to try new age brackets, it is a simple task. However, from a programming standpoint, this is not a trivial task. While experienced SAS programmers understand what a format is, marketers do not, nor are they interested in learning. Therefore, a point and click interface had to be developed which allowed users to drag actual values into the groupings they wished to define. This interface allows the novice to easily regroup selected variables into whatever structure they can need.


Define Range Groups

_B2_ All CP Affin/ Apparel/

Buckets Created Using Format:

Include Bucket	Label	Values
✓	?	-
✓	ACCESSORIES	001
✓	APPAREL	004
✓	CAPS	007
	COACHS SHIRT	008
✓	FLEECE	014
✓	FOOT WEAR	015
✓	GIFTWEAR	016
■	JACKET	216
✓	JERSEY	217
✓	LONG SLEEVE SHIRTS	021

Unbucketed Values
008
216

Hints
Add: Select Last Row
Remove: Uncheck box
Type over to change buckets
Type over or drag and drop
to change values

Lastly, the query engine needed the ability to do analysis and facilitate it in other applications. The client didn't just need a simple query tool with counts; they needed to be able to provide sums, means, maximums and minimums. Result data had to be in a form that could be used by SAS programmers for more sophisticated processes, loadable into Enterprise Miner, and exportable to Excel and HTML as a fully formatted report. Since version 6.12 was the only SAS available and several versions of Excel had to be accounted for, the coding for these tasks fell to the query tool. This is not a trivial task – page breaks are nice to have in a printed report, but not so great in a spreadsheet. As ODS was not yet available, any report created with Proc Tabulate had to be calculated into a data set as well to make it exportable. The result of this requirement was that several different report-generating objects all were subclassed for the creation of different output formats.

QUERYING NON-DATA,

Or how to write a custom query tool with no data. The answer is to use data driven code. Not knowing how your data will be presented provides a challenge, but it is also liberating, in a way. By creating detailed metadata for the application to read, I could provide an engine that was flexible and customizable. Tables and variables could be easily dropped and added from the database by modifying the metadata. Also, by using object oriented design principles, I could write a system where it was easier to add different types of data, simply by adding new classes. The query tool has two main functions linked together: the interface and the business logic. However, by placing the business logic in metadata, we have managed to separate these functions, allowing for the same engine to be used on multiple applications. More complex business logic gets its own class definition, allowing for easy modifications with new business needs. In this way, we have an engine that is both generic and customizable.

DATA-DRIVEN DESIGN

Metadata can mean many things to many people. The Dictionary tables available in SAS in version 6.12 already provide a pretty rich source of metadata. However, this was not enough. With dozens of tables and thousands of variables to choose from, users needed to have a system for narrowing down the list. Once a variable was selected, the user needed to know what values were available for that variable. In different lines of business, the range of available values was different. So, several more tables were added to the metadata read by the interface.

The first was a table to define the different lines of business. With over 4 dozen unique businesses to be tracked independently, there was a need to simply have that list available, along with the appropriate 2 character code used for naming variables. Next was a table to combine these lines of business into logical groupings to allow for easy selection of several at a time. A single business can belong to several groups simultaneously. However the interface had to protect the user from getting the same data twice, since each brand, or logical group, also had it's own table of more summarized information. Without this protection users would end up with reports showing inflated numbers that they were unable to explain.

Also needed were tables for categorizing variables for easy selection. For example, in the first application, there were variables for the last month, last three months, last 6 months, last 12 months, and life to date. These variables maintained a naming convention that allowed for rules to be defined so all the life to date variables could be listed together. By placing this information in a table, the rules are easy to change. Other variables that didn't have a naming convention could still be related using another table in the metadata that simply listed the variable and it's appropriate category. The application also provided a shortcut that users find very attractive. Instead of having to choose the same variable in 4 different lines of business, it only has to be selected once in the interface. The query tool checks the metadata to see if that variable exists for other selected lines of business and automatically selects it for each.

Thus, users can quickly hone in on the variables of interest through the use of categories and then select them with minimal mouse clicks through the use of shortcuts. There is even a table in the metadata pointing to custom wizards for an application requiring specific business logic.

Another useful table in the metadata matched summary tables with applications. This allowed for users to enter into EIS applications that were developed for some of the data, or to create their own reports based on an entirely different mix of variables. Likewise, by tagging summary tables in the metadata, the application is able to utilize the appropriate report generator, which handles summary data differently from detail data.

As the client added complexity to the requirements, more tables were added to the metadata. For example, the original structure called for a single primary identifier for each customer. However, the client had household level data that could be combined with multiple customers. By simply adding a table to the metadata and a new class definition to the application, this was possible. As another example, when the data warehouse was finally available, we discovered that the datamart required 'bucketed' variables, where data is stored in multiple variables, rather than multiple observations. Users want to be able to query all buckets simultaneously, but this requires an OR condition in the query for all the variables. For example, if there are 8 buckets for clothing affinity and I want to select everyone who likes hooded sweatshirts, I have to look at all 8 clothing variables in my query. Better if the interface can figure out how to do that for me. By adding a table to the metadata that defined bucketed variables, the interface is written to look at all the necessary variables automatically. This turned out to be a real bonus for marketing data, as the bucketing issue came up in other applications as well.

OBJECT ORIENTED DESIGN

Rapid application development can not be discussed without object oriented design being mentioned. With 2 months to write a complex application, this tool did not have much of a design phase. However, by creating re-usable classes and utilizing inheritance, the design was expandable and the application was able to perform extremely complex data manipulations. Luckily, with SAS SCL we are able to utilize object-oriented principles in our application. For example, one of the classes used was VARIABLE. The information stored seems simple enough – name, label, format, type, etc. However, that same variable may occur in 12 different lines of business. When a user clicks on it, they want all 12. They may want the sum of all 12, or each individual value from each business. They may also want to use this particular variable as a group variable. All of this information is stored on the object VARIABLE. However, suppose we are working with summary data. This variable already has a weight. It needs to be treated differently than a detail level variable, but many functions overlap. So we create a subclass called VARSUM. This class inherits much from VARIABLE, but it has specific methodology for report production that differs from its parent class. This simple concept extends throughout the entire application. On occasion, there may be a capability needed of the query tool which does not already exist. By using OOP, we can usually reuse our code we've already written and add a subclass for our new twist. So when we decided to allow the user to define a new variable on the fly, we added a new module called VARUSER. It is identical to VARIABLE, except that it provides for the creation code that will create this new variable. Users can even embed another query in VARUSER to create a complex indicator variable. All of this new functionality builds on the capabilities provided by VARIABLE, without affecting it. New complexity doesn't require breaking of old code.

PORTING TO NEW OPERATING SYSTEMS

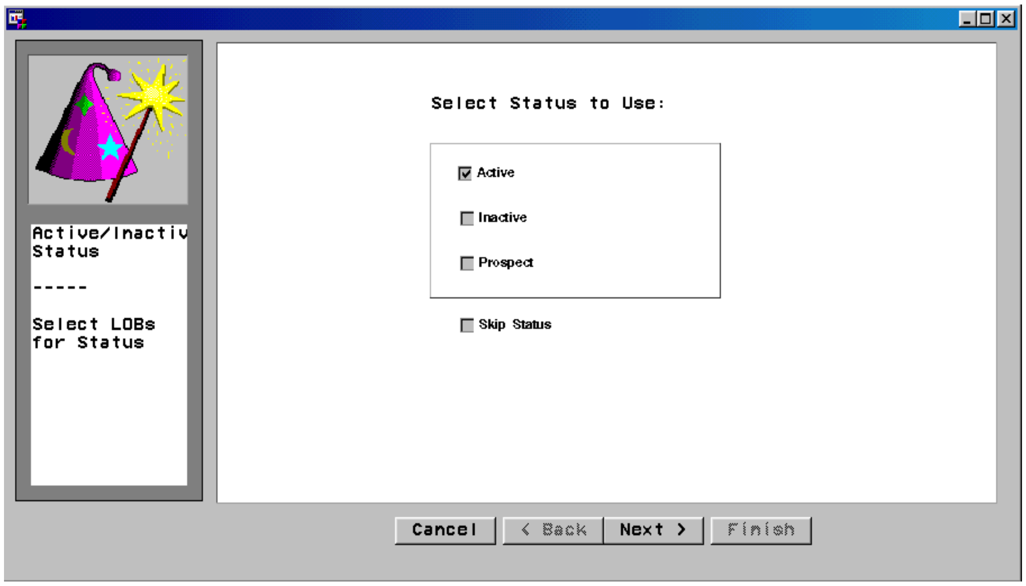
Uncertainty in operating systems is a challenge that SAS has met admirably. During development, the client didn't know where the interface or the data would reside. The result is that the interface can run on Windows (95/98/NT) or Unix, and the data and compute server can reside almost anywhere. Using SAS/Connect, we can generate reports on data on any server with SAS 6.09 or better. Currently there are 10% and 1% samples of the data occupying about 30GB on a Unix box. Users can work out their queries and play with their reports on the sample data with immediate response. The same queries will run in batch on the MVS system against 100% of the data. All of the underlying locations of the data are stored in a table in the metadata that matches the sample with the machine it is stored on. The 100% query is automatically routed to the appropriate machine, and the JCL is inserted as needed.

FEATURES THAT MADE THE OTHER GUYS JEALOUS

Because of the use of object-oriented design, as well as the data-driven nature of the application, some of the features in this query tool are unique. In no other application at the client site is it so easy to create an indicator variable based on a complex subquery. The ability to create ad-hoc groups on continuous (numeric) variables is difficult to find in any query tool – most are predefined by the programmer. By simply creating a subclass and adding an entry into the metadata, programmers can easily add wizards, which guide users through common tasks. The ease of linking this application to an EIS or an Excel application is another custom feature that requires no programming to accomplish a slick-looking transition.

PORTING TO OTHER APPLICATIONS

As it turned out, the same functionality provided by this interface was needed by other applications. Merely by spending a day or two generating metadata, we were able to utilize the same engine



Active/Inactiv
Status

Select LOBs
for Status

Select Status to Use:

[✓] Active
[ ] Inactive
[ ] Prospect

[ ] Skip Status

in three completely different applications. This allowed for the production of an application in a matter of days that could have taken weeks or months. We were able to produce customized query tools in a fraction of the time expected, allowing for more time to be spent on the data, the business logic, and the analysis, as it should be. The design of this interface engine makes it a simple matter to 'drop in' a new data warehouse. It is a simple matter, as well, to customize the interface to the new application. Because of the structure of the objects used, it is relatively easy to replace the screens to give the interface a completely different look. In fact, this same query tool has even been successfully coupled with a Java applet for the interface. The engine itself remained the same, reading the metadata and returning the appropriate results. The interface provided with the tool is sufficient for most needs, and makes for an inexpensive way to surface data to the marketers.

CONCLUSION

Writing a reusable engine has significantly cut back on the coding requirements for our department. No longer are programmers required for every question a marketer may have. SAS has provided the tools necessary to create a great query engine. This tool is extremely powerful in that it is generic, customizable, and easy to use. It may seem like writing such a program is not such a bright idea for a programmer who wants to keep her job. However, just the opposite is true. By producing a program requiring limited resources, the field is cleared for other program development. Our programming needs have actually increased as we are asked to create new applications by more and more groups. Yet, even now, some of those requests are perfectly suited to the engine we already have and we are able to produce a 'new' query tool in a fraction of the time expected.

ACKNOWLEDGMENTS

SAS is a registered trademark or trademark of SAS Institute Inc. in the USA and other countries. ® Indicates registration.

Brand and product names are registered trademarks or trademarks of their respective companies.

Thank you to Charles Bininger, who has agreed to present this paper since the author cannot be present.

CONTACT INFORMATION

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

Amy Swinford
333 S 200 W
Lehi, UT 84043
Work Phone: 801-766-9685
Fax: 801-766-9686
Email: aimless@fiber.net