The Missing Semicolon™

Volume 10,Number 3 Fall 2008

Creating Buckets with SQL

We often have a need to create a table of summarized information for reporting or analytics. That is, one row per key variable/s (i.e. product, customer, order, date, etc) and multiple 'buckets' variables with analytical summarized data. These buckets are usually counts or totals of different events or amounts. In SAS, there are often multiple ways to accomplish the same task. While technically, there is no right and wrong way, our definition of a good program is one that is easy to understand, minimizes CPU and clock time, and has low disk space requirements for permanent and temporary files. In this article, we will take a look at a typical project that requires a dataset with buckets, outline the most common approach, and outline changes to make the program easier to understand and reduce run time and disk space requirements.

The Project

We want to create a dataset with one row per customer and order, with separate amount and counter variables for each product category purchased. Because each order may contain multiple records per product, these rows need to be condensed down to one and separate bucket variables need to be created to keep track of the product purchase amounts and count of items.

The Typical Approach

Let's take a look at how many programmers would attack this problem; keep in mind, this typical approach many not be the most efficient solution.

Like the majority of SAS customers who use SAS/Access to pull data from a third-party database, our fictional company stores it's order data in a third-party database, DB2; so we will be using PROC SQL to access our data. Most jobs at our site use the 'pass-through facility' to connect to DB2, and we have used the same template. Within the sub-query located in the set of parentheses after the 'connection to db2' statement, we pass a SQL query directly to DB2 to join two datasets together from the appropriate subset of data. This DB2 extract is then passed to the outer SAS SQL query where all the columns (select *) are saved in the SAS table 'myExtractFile'.

```
proc sql;
  connect to db2 (db=companyDB
      user=katie using="xxxxxx");
  create table myExtractFile as
   select *
    from connection to db2
     (select cust.custNumber,
         cust.state,o.orderDate,
         o.productCode,o.amount,
         o.promotion
      from sscDB.customers cust,
           sscDB.LineItemOrders o
  where cust.custNumber=
              o.custNumber and
              o.returns ne 'Y');
 disconnect from db2;
quit;
```

Next, the programmer sorts the data to get it ready for the 'set' with a 'by' statement. This step takes quite a bit of time and disk space to run because the extract file is very large.

```
proc sort data=myExtractFile;
  by custNumber state orderDate
     promotion;
run;
```

This last step takes the multiple product records per order and summarizes them to one record. 'Buckets' are created to keep track of total sales amount per product (a_amount, b_amount, c_amount) and total items sold by product (a_unitOrder, b_unitOrder, c_unitOrder). Each bucket is conditionally increased if the order contains the product that corresponds to the buckets being created. Great care is needed to reset the buckets once a new order is encountered. As you see this step is a bit cumbersome to write, and takes an additional pass of the data to process.

Continued on page 3

Systems Seminar Consultants, Inc.
2997 Yarmouth Greenway Drive
Madison, WI 53711
www.sys-seminar.com
train@sys-seminar.com
1-800-997-7081

Live Web SAS Training
Page 8

In This Issue

Creating Buckets with SQL..... Katie Ronk .....Page 1

President's Letter..... Steven First.....Page 2

Quick Tip..... .....Page 2

The SAS SQL Procedure..... .....Page 3

Quick Tip..... ..... Page 4

SAS Enterprise Guide - A Primer..... Rosalind Gusinow.....Page 5

Introduction to SAS Enterprise Guide Class..... ..... Page 5

Efficiencies..... .....Page 6

Custom Training Options ....Page 6

SAS Training at Your Site ....Page 7

Our Open Positions ....Page 7

Live Web SAS Training Schedule..... Jennifer First.....Page 8

Technical Credit and Recognition.....Page 8

Letter From the President

Dear SAS User:

As a SAS user for over 30 years, one of the most important role I have been able to play is making sure that clients are getting the most out of their investment in their SAS licenses. Sometimes we think of suggestions while we are working on a project for a client. But very often, users have us in for the sole purpose of optimizing their investment. So, here are some areas to consider to ensure you are getting the most out of your analytic tools:

1. **Product Assessment-**Do you have the right SAS licenses for your business applications? What else could you be doing with the SAS products you already have licensed? Are there additional products that would pay for themselves and help your business?

2. **Efficiency/Hardware Assessment-**Are you jobs running too slowly? Are you running out of disk space? Are your jobs costing too much to run? Do your jobs fail often? Is your hardware difficult to maintain and administrate?

3. Data Assessment-Is your data 'all over the place'? Are ad hoc queries difficult to construct for end-users? Do even standard reports require considerable effort and detailed knowledge of the database? Is your data not integrated or inconsistent across sources? Does your structure of data mirror business processes or business rules? Does your data model limit which BI tools can be used? Is there a system for maintaining change history or collecting metadata? Is disk space wasted on redundant values? Is data maintenance tedious and ad hoc.

4. Overall process-How do your systems compare to other departments or organizations? Are your processes streamlined? What could you be doing differently? Is it easy to get accurate answers to business questions?

I encourage you to talk with your programming and analytics team about all of the above issues. And feel free to contact us for questions or suggestions. We all make a significant investment in our analytics tools because we value good analytics which help our business. So, let's make sure we are getting the most for our investment.

Steve Firth

run;

SYSTEMS SEMINAR CONSULTANTS, INC.

QUICK TIP

Have you ever needed to set a macro variable based on a logical expression? This is often tricky because of the timing involved in Macro processing. Remember that macro %let's resolve during the compilation phase of processing not during the execution phase. Therefore, they cannot be set will DATA step logic. This is initiated during the execution phase.

Using the IFC and SYSFUNC function makes this easy!

Background information:
IFC – **a SAS function that returns a character value based on whether an expression is true, false, or missing.**
IFC(*logical-expression*,
value-returned-when-true,
value-returned-when-false,
value-returned-when-missing)

SYSFUNC - **Execute SAS functions**
%SYSFUNC(function(argument-1,...argument-n))

Conditionally Set a Macro Variable

Problem:
We want to set the macro-variable value of 'MstartDt' if it is not already set. In other words, if 'MstartDt' is missing we want to set the value to the beginning of the year - '01JAN2008'd. If 'MstartDt' is NOT missing we want to set the value to itself (&MstartDt).

The ifc function will perform the logic check:

```
ifc(&MstartDt =
%str(.),'01JAN2008'd,&MstartDt)
```

1) Check the condition in the first argument:
   Is MstartDt missing?
2) Is this true?
   If so, return the value of the 2nd argument.
3) Is this false?
   If not, return the value of the 3rd argument.

The sysfunc function will execute the ifc function:

```
%sysfunc(function() );
```

Put it all together:

```
%Let startDt =
%sysfunc(
ifc(&MstartDt =
%str(.),'01JAN2008'd,&MstartDt)
        );
```

run;

Creating Buckets with SQL

CONTINUED FROM PAGE 1

```
data myExtractFile2;
   set myExtractFile;
   by custNumber state orderDate promotion;
   retain a_amount    b_amount     c_amount
          a_UnitOrder b_UnitOrder  c_UnitOrder 0;
   if productCode = 'A' then
     do;
        a_amount + amount;
        a_UnitOrder + 1;
     end;
   else if productCode = 'B' then
     do;
        b_amount + amount;
        b_UnitOrder + 1;
     end;
   else if productCode = 'C' then
     do;
        c_amount + amount;
        c_UnitOrder + 1;
     end;
   if last.promotion then
    do;
        output;
        a_amount=0;
        b_amount=0;
        c_amount=0;
        a_unitOrder=0;
        b_unitOrder=0;
        c_unitOrder=0;
    end;
run;
```

We now have the dataset in the format we need to use for reports and analysis. However, this program takes a long time to run and we want to explore a more straightforward solution.

Doing More with SQL

The data step used above is somewhat complex and at first glance doesn't intuitively translate to SQL. But what is being done in the data step? We are creating new product variables that contain the costs or counts for each order and writing out only one observation per order. Translated to SQL speak, we are GROUPing our data by the variables that define a unique order and conditionally adding dollars or counts to a new variable depending on the product purchased (case-when logic with a summary function). So, we need to:

1) Add a 'GROUP by' statement listing the same variables we had on our data step 'by' statement: `group by custNumber, state, orderDate, promotion` When using the pass-through facility, the 'group by' statement could go on the inner or the outer query. We are choosing to add it to the outer query so that SAS performs the summarization, though this may or may not be the most efficient place depending on the client environment.

2) Explicitly reference the 'by' variables on the outer select clause: `select custNumber, state, orderDate,..`

3) Add summary functions with embedded case-when logic to conditionally add up our buckets:

```
sum (case when productCode='A'
          then amount
          else 0
     end ) as a_amount,
```

The program below puts all this logic together to create the bucket variables in the PROC SQL step.

```
proc sql;
   connect to db2 (db=companyDB
      user=katie using="xxxxxx");
  create table myExtractFile_NEW as
  select custNumber, state, orderDate, promotion,
           sum(case when productCode='A'
                    then amount
                    else 0
               end) as a_amount,
           sum(case when productCode='A'
                    then 1
                    else 0
               end) as a_unitOrder,
           sum(case when productCode='B'
                    then amount
                    else 0
                end) as b_amount,
           sum(case when productCode='B'
                    then 1
                   else 0
               end) as b_unitOrder,
          sum(case when productCode='C'
                    then amount
                    else 0
               end) as c_amount,
          sum(case when productCode='C'
                    then 1
                    else 0
               end) as c_unitOrder
   from connection to db2
        (select cust.custNumber, cust.state,
                o.orderDate,     o.productCode,
                o.amount,        o.promotion
        from sscDB.customers        cust,
             sscDB.LineItemOrders   o
         where cust.custNumber=o.custNumber and
               o.returns ne 'Y')
   group by custNumber,state,orderDate,promotion;
   disconnect from db2;
quit;
```

Continued on next page

Results

The all-in-one SQL step cut processing time in half; most of the time was saved by eliminating the need for the 'PROC SORT". Results will vary depending on your data and platform configuration. Further efficiencies might be gained by passing the query and logic to the inner query (DB2) and/or using the DB2 LIBNAME engine instead of the pass-through facility.

Further Efficiencies Added

Let's try adding a few more efficiencies. We are now using the LIBNAME engine to set up a library (SASDB2) to point to a set of DB2 tables. When we reference these tables by name, we can write code as if we were working with SAS tables. Behind the scenes, SAS will translate our code to DB2 SQL queries. The LIBNAME engine usually performs optimally when the SAS code is written in PROC SQL and SAS specific functions are kept to a minimum in the code. Notice, we no longer need the inner query that goes to DB2. SAS will now take care of this for us.

```
libname SASDB2 db2 db=companyDB user=katie
                   using=xxxxxx schema=sscDB;
proc sql;
   create table myExtractFile_NEW2 as
   select cust.custNumber, cust.state,
          orders.orderDate length=4,
          orders.promotion,
          sum(case when productCode='A'
                   then amount
                   else 0
              end) as a_amount,
          sum(case when productCode='A'
                  then 1
                  else 0
              end) as a_unitOrder length=4,
          sum(case when productCode='B'
                   then amount
                   else 0
              end) as b_amount,
          sum(case when productCode='B'
                   then 1
                   else 0
              end) as b_unitOrder length=4,
          sum(case when productCode='C'
                   then amount
                   else 0
              end) as C_amount,
          sum(case when productCode='C'
                   then 1
                   else 0
              end) as C_unitOrder    length=4
   from   SASDB2.customers cust,
          SASDB2.LineItemOrders   orders
   where cust.custNumber=orders.custNumber and
         orders.returns ne 'Y'
  group by cust.custNumber, state,
           orderDate, promotion;
quit;
```

Also note, that we added 'length' options to reduce our file size. By default, SAS assigns numeric variables a length of eight. Eight bytes are more than double the bytes required to store most integers. Most counts and date fields can safety be stored in three or four bytes, depending on your platform. Reducing the width of your dataset not only saves disk space, it will reduce the time for I/O therefore reducing overall program run time.

Conclusion

When creating a dataset with buckets, keep the SQL group by and case-when logic in mind: it could potentially greatly simplify your process. For general efficiencies, reduce variable lengths whenever possible and cautiously utilize the LIBNAME engine, instead of the Pass-Through Facility for accessing Third-Party databases.

run;

QUICK TIP

There are times when we need to create .dbf files because another application needs them in that format, or we need to read .dbf files because another application created them for us to use.

If you want to use a .dbf file, use SAS/Access to OLEDB with a LIBNAME statement.

Define the Location of the Data Source

Define the data source as the directory where the .dbf files do or will reside, using the following syntax:

```
/*   Use and Create dbf files   */
/*   DATA SOURCE is c:\temp     */

LIBNAME dbflib oledb init_string=
"Provider=Microsoft.Jet.OLEDB.4.0;
 DATA SOURCE = c:\temp;Extended
Properties='dbase IV'
";
```

Use the Data Location as a Source and Destination

You can then reference the .dbf file specified in the Libname statement above, in a Data Step as follows:

```
/*create temporary SAS dataset 'map' */
/*from c:\temp\usmap.dbf               */
data map;
    set dbflib.usmap;
    where latitiude between 44.16 and
50.3;
run;

.
data map2;
   set map;
. . .
run;

/*create c:\temp\newmap.dbf             */
/*from temporary SAS dataset 'map2'     */

data dbflib.newmap;
   set map2;
run;
```

run;

SAS Enterprise Guide - A Primer

As SAS professionals, we have spent a lot of time honing our skills – making sure that every keyword was corrected and every semi-colon was dotted. Then what happens – along comes some new SAS product that is supposed to make our life and the like of non-SAS professionals easier. We all have a sense of what that means – another new-fangled tool to make life more complicated, something we'll use but only if we're dragged to our keyboards kicking and screaming. I'm here to tell you – it's not that bad, in fact it's actually pretty neat!

The main advantage is its point-and-click ability.

- We can use it to select any type of input data and create a SAS data set.
- We use it to select a task we want to perform, identify the variables and the way we want to use them.
- We use it to modify options and rerun the task
- We can use it to view the generated SAS code as a base for traditional methods.

All in all, there is less memorization involved. All I have to do is define my SAS data sets.

- If I know the file I want to use, EG knows the variable names.
- If I want to perform a task, EG knows the procedure to use.
- If I need to make a change, I can play with all types of 'what-if' scenarios and can save and store the steps as distinct entities.
- If I want to perform the task in a traditional manner, EG has the SAS code I can use.

It might be frustrating at first to create your SAS data sets from non-SAS files. It might be frustrating at first to figure out which task is associated with which SAS procedure. Wasn't this true when we were first learning traditional SAS methods? Don't lose sight of the fact that you only have to define your files once. Don't lose sight of the fact that it doesn't really matter what procedure is run behind the scenes, what matters is that EG produces the accurate report results you're expecting.

I will start this series of articles discussing the methods of analyzing information using our traditional SAS methods and then using SAS Enterprise Guide.

The topics for this article include:

- Creating a SAS dataset from a non-SAS file or a SAS data set
- Analyzing the information in a SAS data set – counts and summaries

Our assignment for today is to take a non-SAS data set on our local computer (C:\Newsletter\EG\Sasi0301.txt) and create a SAS dataset that we will analyze by looking at both the detail and summarized statistics.

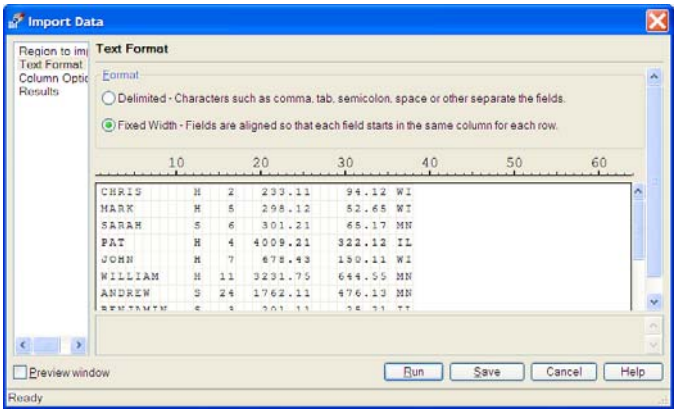
Traditional SAS Method-External Flat File

Using traditional SAS methods means creating a job which uses a Data step and a Proc step to look at the detail. In order to accomplish this we have to find someone who knows the file location and the file layout. We have to manually type in the location and the variable attributes – give each one a name, type (the default is numeric), starting and ending location or informat, format, label, etc..

```
DATA  softsale;
  infile  "C:\Newsletter\EG\Sasi0301.txt";
  input   Name       $1-10
          Division  $12
          Years      15-16
          Sales      19-25
          Expense    28-34
          State     $36-37;
RUN;
```

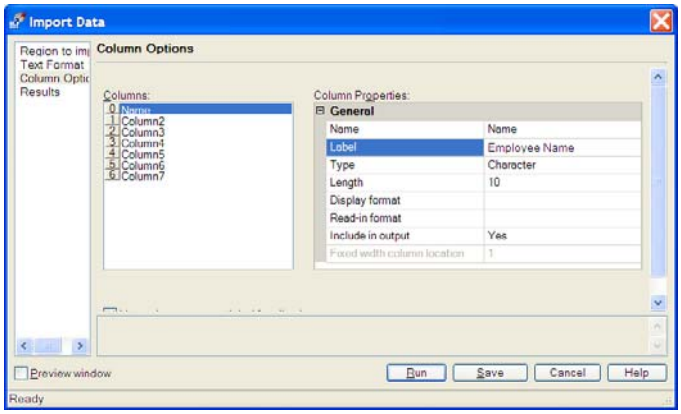
SAS Enterprise Guide Method—External Flat File

Enterprise Guide wizard style approach to importing flat files simplifies the process of reading in large fixed width files; however, care needs to be taken to make sure variable attributes are set up correctly. The main advantage of the import feature for fixed width files is the sneak peak at your data and drag and drop bars to separate out columns.



Continued on next page

If the variable names are not the column headings, the names need to be entered manually in a separate tab called *Column Options*. Variable attributes are set in the *Column Options* tab; all variables default to character and need to be changed to numeric if applicable. Many new Enterprise Guide users forget to do this and problems are caused later when these variables are not in the right format to be summarized, etc. Though the import data window is not necessarily intuitive, it's easy enough to pick up after a simple lesson and can save time coding.



A brand new SAS user would more than likely find this much easier to learn than writing an INPUT statement. A SAS old-timer might find EG easier at times, though many of us still have a tendency for manually writing a quick data step for those simple file imports instead.

A complete step-by-step guide to importing this flat file will be put on our website. Now let's look at summarizing our data.

Traditional SAS Method- Summary Table

We will use a Proc Summary Step to generate a dataset containing summarized statistic for each state in. We will use Proc Print to create a summarized report from this dataset. Decide what variables you want to analyze, what statistics are important … and don't forget the syntax needed by Proc Summary, in particular the Class and Output statements. Perhaps a Proc Sort step will be helpful to arrange the observations so that the totals appear last – don't forget the descending option.

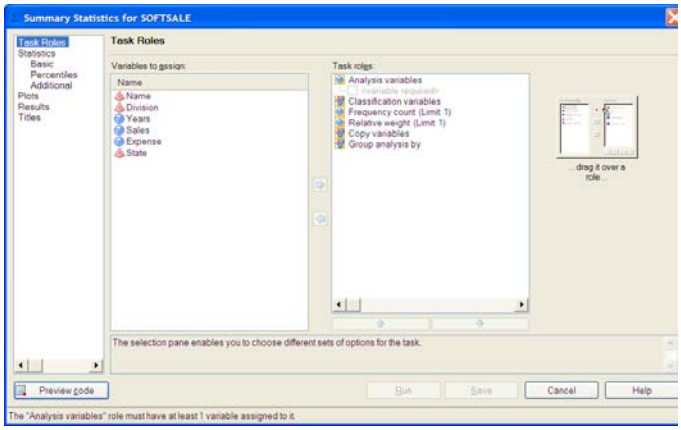

```
PROC SUMMARY data=softsale;
   CLASS state;
   OUTPUT out          = softsale_statistics
        sum(sales)     = sum_sales
        sum(expense)   = sum_expense
        mean(sales)    = mean_sales
        mean(expense)  = mean_expense
        n(sales)       = n_sales
        n(expense)     = n_expense  ;
RUN;
PROC SORT data=softsale_statistics;
   BY descending _type_ state;
RUN;

PROC PRINT data=softsale_statistics
                              (drop=_type_ _freq_);
   title
  "Summarized Softsale Dataset - classed by state";
RUN;
```

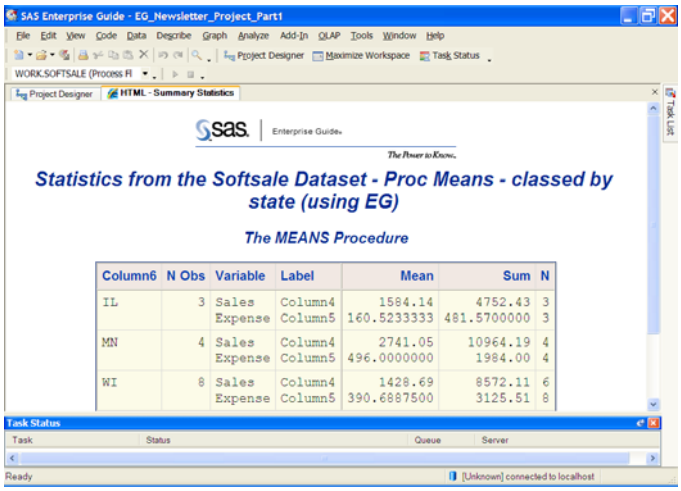
SAS Enterprise Guide Method- Summary Table

One of the great things about SAS is the ability to efficiently summarize large volumes of data using PROC SUMMARY or PROC SQL. However, the syntax for both of these procedures is a bit cumbersome, and it is not uncommon for it to take more than one or two attempts before getting the syntax and results correct. The Enterprise Guide Summary Statistics wizard allows you to drag and drop your analysis and classification variables. Now, not only does one not need to remember the syntax correctly (along with typing it correctly), but the names of the available variables are there for the user to see.

Continued on next page

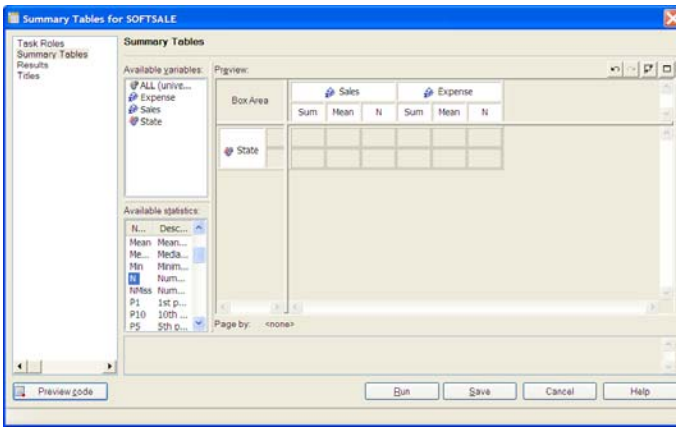


Options are available to create a report 'proc means' style or create a permanent dataset. It might take a while to remember the location of the appropriate options, but you will become a pro in no time.



SAS Enterprise Guide Method—Tabular Report

For the average base SAS programmer, tabular reports have traditionally been challenging to create. Rare was the programmer that did not need to reference some form of documentation while coding a complicated table statement. One of the highlights of Enterprise Guide is the window for dragging and dropping variables into a tabular report. Keep in mind this code can be cut and pasted into a traditional SAS program.



Conclusion

I hope this article demonstrated some of the many advantages of SAS Enterprise Guide. In future issues we will explore creating parameters in Enterprise Guide, creating SAS stored processes and using the SAS add-in feature in Excel to execute them. A step-by-step guide to this Enterprise Guide example can be found on our website.

Public or Live Web Training

Learn from the Comfort of Your Office or Join Us at Ours!

Over 26 Years of SAS Training Experience

Complimentary Follow Up Help Desk

Our Trainers Are Also Expert Consultants

Introduction to SAS®
- January 20-22
- April 13-15
- June 15-17

SAS® Enterprise Guide
- January 15-16
- April 16-17
- June 18-19

Advanced SAS
- February 2-4
- May 4-6

SAS® SQL Procedure
- February 26
- May 21

Exploiting SAS® ODS
- February 5-6
- May 7-8

SAS® Macros
- February 23-24
- May 18-19

SAS® Efficiencies
- February 27
- May 22

Advanced SAS® Macros
- February 25
- May 20

SAS® Report Writing
- March 16-17
- June 8-9

Tips, Tricks, & SAS® Techniques
- March 19-20
- June 11-12

PROC Report
- March 18
- June 10

For questions or registration, contact
Our Training Coordinator at 1-800-997-7081, ext. 306
or visit www.sys-seminar.com

Steve First
President

Katie Ronk
Director of Operations

Andrew First
Consultant

Rosalind Gusinow
Senior Consultant/Trainer

Jennifer First
Office Manager

Dan Fischer
Consultant/Trainer