Paper 032-29

Practical application of SAS® for the beginner - enabling the replacement of a fragile Access

David Yarmchuk, Canadian Pacific Railway, Calgary, AB, Canada

ABSTRACT

In an attempt to find a method to create a stable, fast, easily modified replacement for a Microsoft Access database, SAS rapidly became the only choice as a replacement tool. SAS can very quickly be adapted for uses by relative beginners in novel ways to exploit its strengths as a data extraction and manipulation tool. Using PC SAS and SAS CONNECT for ODBC a method was found to convert the most fragile parts of a Rail Car Setoff reporting database. Aimed at the well-informed MS Office user with either exposure to or an interest in SAS, this paper is intended to introduce PC SAS as a tool that does more than just statistical analysis but also is a viable alternate tool to the more traditional programming solutions. This shift from MS Access to SAS cut processing time from over 20 minutes to less than 4 and reduced the complexity of the database from multiple nested macros down to a single file that is in plain text in easily readable SAS® code.

INTRODUCTION

In many companies today IT departments struggle to complete their large and medium sized projects, balancing budgets, timelines, scope and available talent. This leaves many business departments without the access to IT resources for the development of their smaller data gathering and reporting systems. When the IT resources are available, the cost to engage them can be prohibitive. Typically, IT departments also have rigid policies and procedures geared towards larger projects that do not scale down for use with small projects. This leads many business units to create their own solutions and in some cases their own systems teams. Often the tools selected are based on the productivity tool suite used by the business unit. In many cases, the tool of choice is Microsoft's Access. Microsoft Access is an excellent tool for database solutions; it is both flexible and robust. Where these internally crafted solutions fall down is not in the initial design or implementation, but in the slow modification and growth of the solutions. Canadian Pacific Railway has many examples of these systems evolving from a simple solution to a complicated program supporting a large number of processes. The best way to deal with these products of software evolutions is to either replace the system with a completely new solution in the same language/tool or replace key parts with a tighter more reliable language/tools.

SAS® is an excellent tool to replace either the entire system or key parts of it. Making changes like this can lead to more stability, better performance, and can simplify internal structure of the system[1].
Faced with the challenge of maintaining a legacy Access database that is key to CPR's business a decision was made to re-write a number of key sections. This database had passed through the hands of a number of business developers in several different departments leaving the database complicated and fragile. On reviewing the issues, the most fragile section seemed to be the macros and queries used to collect and manipulate the data weekly and monthly. Two options were open at that point, 1) Rewrite the process using VBA and redevelop the queries for stability or 2) Re-develop the process in a different tool. Due to the complicated relationships inside the Access database, it seemed simpler to develop a new reporting process[2]. As PC-SAS is a preferred tool for the department, the redevelopment was a natural for this work. Selecting PC-SAS enabled the redevelopment of the extraction routines and then synthesis of the data in to its end report ready state. This was a daunting task.

First, the existing processing structure needed to be modeled. This provided the opportunity to slowly review each of the nested macros to identify all of the steps taken as well as identify each of the queries executed. Once the steps were documented, the queries needed to be reviewed. As queries can be built up of both tables and queries it was critical to take the time to identify all of the dependencies for each query and to understand what each sub-query or table adds[3] to the overall query. Once the data was understood and its manipulation mapped it needed to be extracted from the Access database and then processed. Once finished with the processing the

[1] Due to the natural language used in SAS and its linear nature it can be very easy to follow the data processing; MS Access can next query after query obfuscating the data source and its makeup.
[2] The data capture process is also in the database but is stable and so was not considered for development.
[3] Or doesn't add as the case may be. There are times where tables are included in a query without any benefit.

data was exported for insertion in to a Microsoft Excel spreadsheet[4].

II – MAPPING THE MACROS

Mapping the overall process is a key step in understanding an Access database. Imagine all of the quick fixes, updates, modifications, and new development that are part of the program. Without a clear understanding of how all steps relate there are many opportunities for error or omission, additionally there is a chance to miss opportunities to prune out pieces of macros that are not needed anymore. The listing of macros can be very long and complicated making wading through all of the code very difficult. The best way to begin mapping is to first find the main macros or the first one in the series that needs to be run, for example what form starts the process or what macro needs to be run first. If the program is complicated, you may run across a macro with a list of macros as its tasks rather than actual tasks. Expect that you will need to look at each sub-macro in turn to understand the whole process. Although the names of the macros can be a help to determine what each macro does, there is no reason to believe that the name is a true reflection of its function. As you go through each macro please note the order that data is manipulated and what the query names are. This order will be priceless when you begin to pick apart the queries.

You might not have macros; you might need to read and document VBA[5]. The requirements are the same but VBA can be easier or more difficult depending on the skills of the programmer who last worked on the code.

Regardless of which method is used to string together the process for manipulating and producing data the act of documenting the process will provide detailed insight into the architecture of the existing solution.

III – DOCUMENTING THE QUERIES

Once it was known how the solution was to be strung together, understand how the data is manipulated was needed. There are a number of ways that an Access database can manipulate data for reporting purposes.

The method ran across most often is born out of the inability of Access to handle very complicated queries. These queries involve multiple levels of summarized and non-summarized data, when Access tries to execute them it raises an error identifying the query is too complicated. When the error is encountered the help system recommend the creation of a temporary table to hold intermediate results for use in later queries. This breaks up the complicated query into something that Access can handle[6]. The side effect of breaking the queries up is that they need to be run one after the other. Serial execution means that either the user of the system runs each query in order or the developer "automates" the process with VBA or Macros.

The alternative method typically is when the queries are not too complicated. This means that they can be executed without errors. Because the execution does not fail, there is no need for intermediate tables. Since there is no need for a series of steps to manipulate the data there is far less pressure to automate the query so the database is usually less complicated and easier to document.

Regardless of the complexity of the relationships of queries and tables, it is important to understand what each query contributes to the final product. The best approach is to look at each query individually to build up a picture of the overall manipulation of data and then formulate as simple a description of what is done to create the result. You need to do this to ensure you are not locked into an Access mindset. It is easy to approach to manipulation of the data in the exact same manner as Access does, even if there are many opportunities to optimize by using the enhanced capabilities of SAS.

In this case, the database first mapped out the macros and then the queries. There were several streams of queries each with their own result set. Because of this each result set needed to be approached as a separate task. Having mapped out each of the tasks, the replacement began to be developed in SAS. It was first approached with the intention of exactly replicating the current method. This meant an immense number of intermediate datasets and lots of sorting and merging. After having replicated the entire first stream, the approach

[4] The data was placed in Excel to allow other departments to access the data using their own tools for inclusion in their reporting processes.
[5] Visual Basic for Applications
[6] Multiple Stages, each stage a small bite of the overall outcome required. Queries feed tables that are then fed to more queries. Ironically this mirrors SAS® execution stream.

began to be questionable and it became obvious that the application of PROC SUMMARY with class variable would allow enumeration of all the possible values for each of the class variable. All that would need to be done then was to select the rows out of the result set. The Result was that rather than using an 18 step process that was marginally faster than the original it was shortened to a 3 step process that was easily 5 times faster than the original and at least twice as fast as the first SAS® attempt.

CASE: The macros were made up of a series of steps repeated over and over. First the intermediate table would be deleted, then the first query would be run averaging the hours to repair, hours to lift[7], and set off hours[8] as well as producing a car count. The data would be filtered to ensure that only Cars with a traffic flag, falling over a specific date range[9], that were not reported damaged while empty and not considered priority[10] would be analyzed. Then the next step was to measure the same values for a different breakdown of the traffic type. In the second query the traffic was broken down as loaded or empty traffic, for a specific date range where it was of non-intermodal[11] type in the previous traffic flag and it was not reported damaged while empty[12]. Which means there is a total breakdown of damaged car setoffs by intermodal vs. other traffic, and then a further breakdown of other traffic into loaded vs. empty traffic. The next step is to the break this down by different conditions to get a finer view of the way damaged car are being handled in all field circumstances these breakdowns can be those set off in terminals[13] or those setoff enroute[14]. Each of the 3 previously listed results involves the deletion of a table, the insertion of one queries data and then the insertion of another queries data which is 3 steps per data grouping. Of course, then the same groupings are enumerated for priority traffic (rather than the non-priority traffic identified earlier) leading to a further three sets of results each of which in turn has 3 steps to produce them. This ends up with us having 18 steps with much deletion of Microsoft Access objects[15].

IV – ACCESSING THE DATA

The Access to ODBC module was used to pull data from the Access database. The performance was good enough to download the entire table to a temporary dataset while the results were produced. The following code is all that was necessary to produce a dataset for the SETOFFS table:

```
PROC SQL;
CONNECT TO ODBC(dsn="DSN=MS Access
Database;DBQ=R:\CGY_GCS\CAR_PLANNING\setoff\Setoff2000fe.mdb;DefaultDir=R:\CGY_GCS\CAR
_PLANNING\setoff;DriverId=25;FIL=MS
Access;MaxBufferSize=2048;PageTimeout=5;UID=admin;");
CREATE TABLE WORK.SETOFFDATA
AS SELECT * FROM CONNECTION TO ODBC
(       SELECT *
        from [SETOFF]);
disconnect from ODBC;
```

The DSN acts like a pointer to the database so that ODBC can find and access the Database. Once the data was

[7] Hours to lift – the number of hours it took to pull the car from a siding, where it was set off when it was reported damaged.
[8] The set off hours is the amount of time the car sat on a siding waiting to be repaired and lifted back in to service.
[9] The date range is one of 1 week from the previous Friday to the end of the current Thursday.
[10] Cars considered priority are measured next.
[11] Intermodal is container traffic. This traffic is usually considered very time dependant and so is considered priority traffic.
[12] Even though the car was not reported damaged while empty (it is loaded) the car may be able to continue to destination and be unloaded before it is repaired.
[13] Terminal setoffs are those that occur as a train passes through a rail yard but was not scheduled to stop there, but does to setoff a car.
[14] Enroute setoffs are those that occur outside of a rail yard.
[15] Repeated deletion of Microsoft Access objects can be a bad thing, typically it causes the database to rapidly bloat from keeping track of all of the changes and may cause unexpected effects in the database leading to instability and failure.

loaded it was able to perform further data manipulation on the mainframe[16] to produce classifications[17] to assist in identifying if the traffic is intermodal or not.

An interesting side note is that the above code can be used in any circumstance you are accessing external data through ODBC. All that is necessary is to change the DSN reference, the output dataset and lastly the SQL in the connection to ODBC sub-query. With this tool, you can access any data source that is accessible from ODBC including databases from ORACLE to RDB.

Once the data is collected and classified, the datasets can be produced to resemble the Access tables.

V – PROCESSING THE DATA

The first attempt to produce the required data resulted in an almost exact replication of the process used in Access. This meant that for every step used in Access there was a step in PC SAS®. Although PC SAS® outperformed Access even when performing the same steps in the same order it became obvious very quickly that there was another opportunity to exploit one of the strengths of SAS®. To produce the averages necessary the code was changed to use the PROC SUMMARY procedure. The documentation mentioned that there was something called class variables that could be used to enumerate the averages for all possible values of the class variables. This meant that if there was one class variable that had two possible values there would be three results, an ALL result and averages for RESULT 1 and for RESULT 2. Several PROC SUMMARY steps were built that would allow extraction of the relevant information needed to build each result grouping. This building up meant that it was about three steps in SAS® for six steps in Access. This was getting better but it developed there was real potential to include all of the variables that might be needed to filter on. In this way, SAS® would produce a large result set but it could then filter for all possibilities. This meant that all 36 steps[18] could be easily captured in seven steps[19] resulting in a 5x savings in processing time.

```
PROC SUMMARY DATA=SETOFF.SETOFFIMDLVSOTHER;
    CLASS FLAG BOEMPTY PRIORITY LE TERMINALCODE;
    VAR REPAIR_HOURS HOURS_TO_LIFT TOTAL_SETOFF_HOURS;
    OUTPUT OUT=SETOFF.REGULAR_SETOFFS MEAN=AVG_REPAIR_HOURS AVG_HOURS_TO_LIFT
AVG_TOTAL_SETOFF_HOURS;
    WHERE LIFTDATE >= &STARTDATE AND LIFTDATE < &ENDDATE;
RUN;

*** EXTRACT REGULAR SETOFFS;
DATA SETOFF.IMS_OTHER_LOAD_EMPTY;
    SET SETOFF.REGULAR_SETOFFS;
    IF BOEMPTY = 0;
    IF PRIORITY = 0;
    IF FLAG = 'INTERMODAL' OR FLAG = 'OTHER';
    IF FLAG = 'INTERMODAL' AND LE THEN DELETE;
    IF TERMINALCODE = 'NO' OR TERMINALCODE = 'YES' THEN DELETE;
    IF LE = 2 THEN FLAG = 'EMPTY';
    IF LE = 1 THEN FLAG = 'LOAD';
RUN;
…
```

The PROC SUMMARY statement takes a relatively long time to execute[20] but once it was complete, the extraction steps were very quick. All the querying that Access had spread over the entire macro could be completed in parallel in the PROC SUMARY. Instead of producing than 6 or 12 smaller result sets, one large result set would be

[16] Please refer to appendix I for a sample of the code for this. Note I will not describe how to create a connection to your MF.
[17] These classifications are from the industry reference file we maintain on our mainframe called UMLER (Universal Machine Language Equipment Register).
[18] That's 18 steps for non-priority and 18 steps for priority cars
[19] 1 step to produce the main dataset and 6 steps to produce each of the individual result sets.
[20] Compared to any one step in the Access Macro

created in this way only the rows in the dataset that meet each of the criteria would be extracted to fill each dataset. Once extracted, the data can be easily fed to Excel spreadsheets using the ODS HTML/EXCEL method[21]. Once the data is in Excel, it can feed the data into an existing process, redevelop the process or create a new process to transfer the data from Excel into a formatted report. That goes beyond the scope of this paper; there are many good-sized books on Excel and VBA available to determine how to manipulate the data.

CONCLUSION

Although SAS® has not completely replaced the entire Access database, the fragile portions of the program were replaced. SAS® provided the platform to transform complex data from multiple sources using a more reliable, stable, simple, and faster method. The ability to quickly integrate SAS into an MS Office environment cannot be underestimated, in any shop where there are MS Office tools being used to manipulate data there is a huge potential for replacement and enhancement.

[21] Using the Output Delivery System you can export to html with the extension as .xls Excel will interpret it as an Excel spreadsheet rather than HTML which can then be used as a data source for building a final series of reports in Excel.

APPENDIX I

Code sample for mainframe processing:

```
DATA WORK.SETOFFDATA;
  SET WORK.SETOFFDATAINITIAL;
  CARID=CARNUMBER;
RUN;

DATA SETOFF.CARLIST(KEEP=CARID);
  SET WORK.SETOFFDATA;
RUN;

RSUBMIT;

*** UPLOADS A SAS DSN TO THE MAINFRAME ***;
LIBNAME SETOFF 'UMEC.CAR.SETOFF.DATASETS';
PROC UPLOAD DATA=SETOFF.CARLIST OUT=SETOFF.CARLIST;
RUN;

*** PROCESSES SAS / MAINFRAME DATA ON THE MAINFRAME ***;
FILENAME UMLER 'UMEC.CARAVAIL.UMLER' DISP=SHR;

DATA UMLER;
 INFILE UMLER;
  INPUT @01  CARID   $CHAR10.
        @27  RPSCODE $CHAR02.
        @29  NEWCODE $CHAR10.
        ;

PROC SORT DATA=UMLER;
  BY CARID;

PROC SORT DATA=SETOFF.CARLIST;
  BY CARID;

DATA NEW;
  MERGE SETOFF.CARLIST (IN=A) UMLER;
    BY CARID;
      IF A;

*** DOWNLOAD SAS DATASET FROM THE MAINFRAME ***;

PROC DOWNLOAD DATA=NEW OUT=SETOFF.CARRPS;
RUN;

ENDRSUBMIT;
RUN;
```

CONTACT INFORMATION

Your comments and questions are valued and encouraged.

Contact the author at:

David Yarmchuk
Canadian Pacific Railway
Suite 500 401 9th Ave SW
Calgary, Alberta, Canada
T2P 4Z4
Work Phone: (403) 319-6886
Email: dave_Yarmchuk@cpr.ca
Web: http://www.cpr.ca