Paper 170-28

Why SAS® is the Best Place to Put Your Clinical Data

Steven A. Wilson, MAJARO InfoSystems, Inc., Santa Clara CA

Abstract

The SAS System contains an incredible amount of functionality for managing your data files as a database. In most respects, including data storage, data access, and data management, SAS can be considered a database.

This paper will review the database functionality available within the SAS System and how this relates to the task of clinical trails data management. Sample code, which illustrates many of these database techniques, is provided.

Introduction

Working in an industry that is highly regulated by a large federal entity, the FDA, presents unique challenges in data management. We need to quickly take advantage of new technologies and capabilities that become available.

Contrary to popular misconceptions, the SAS System *does* provide the functionality and flexibility that is required of a clinical database system. The next few pages overviews SAS System features that provide the functionality that is required of a DBMS. Detailed discussions of each of these topics are easily obtained in SAS documentation or SUGI presentations.

Why SAS is the Answer

Clinical data is sensitive information that should be treated as a valuable asset as well as a regulated commodity. Not only can improper management of clinical data cause the invalidation of an entire clinical trial, but it may also result in strict remedies, including serious financial penalties, from the FDA.

To avoid such costly outcomes, the FDA requires CFR 21 Part 11 compliance for a clinical database. This requires that all clinical DBMS prevent unauthorized access to data and maintain complete audit trails.

Data must be password protected and allow updates only by authorized users. The FDA requires that entry and revision of clinical data be logged to indicate the user performing the action, the date and time of the action, as well as the reason for change of data.

These requirements are all easy goals when working with the SAS System.

The FDA has selected SAS transport data sets as the standard for receiving electronically submitted data. Since the FDA accepts submissions electronically and that FDA reviewers are equipped with the SAS System Viewer, keeping clinical data in SAS makes sense. The use of SAS is essential to clinical information management.

Analysis for a submission is typically done in SAS and reporting is also effectively accomplished using SAS. So why is data not always gathered and stored using SAS?

Answers to that question have relied on the invalid belief that SAS is not a database. This paper will illustrate that this is not a correct assumption.

Is SAS a Database?

Technically, a collection of data is a database. Is a text file a database? Those who perform text mining may have an answer that differs from database purists.

What people really want is a clinical Database Management System (DBMS) – a database with advanced functionality. A database is only one component of a DBMS

Common expectations of a DBMS include:

- Long variable lengths
- Data compression
- Generation data tables
- Indexing
- Data entry integrity
- Integrity constraints
- Referential integrity constraints
- Audit trails
- Password protection of data tables
- Encryption of data in tables
- Encryption of network traffic
- Data views
- Query tools and programming interfaces
- Query optimization
- Row-level locking for concurrent data access
- Reliability
- Performance and scalability
- Data integration, warehousing and mining

This DBMS functionality should assist with speedy data management and ensure the security and auditability of the data.

Various reasons have been suggested for not using SAS as a clinical DBMS. One fallacy is that SAS is not a Relational DBMS.

In the relational data model introduced by Dr. E.F. Codd in 1970, data and relations between them are organized in tables, allow the definition of integrity constraints, and can be manipulated using relational algebra and SQL. As this paper will show, it can be argued that SAS is indeed a DBMS because it can control the organization, security, storage and retrieval of data via the relational data model.

Other reasons concern the available clinical software. In-house systems have been deemed too expensive to develop and maintain. Thus, many companies have migrated to off-the-shelf products. Unfortunately, most of these store your data in non-SAS data structures.

This has placed many companies in the unseemly and costly situation of having to maintain both a DBMS software system *and* their SAS-based analysis and reporting applications, moving data between the different software platforms as needed.

SAS is already running in 90% of the Fortune 500, so why maintain a separate DBMS?

An examination of the conditions that exist in this industry as well as the available software would lead to the conclusion that we require a SAS-based clinical DBMS solution.

MAJARO InfoSystems has offered such a solution since 1987. ClinAccess is an off-the-shelf SAS-based solution for clinical trial data entry, management, and review that keeps clinical data in SAS from start to finish. Our development efforts center around the assumption that SAS is uniquely qualified to be the ideal clinical DBMS.

The remainder of this paper gives an overview of DBMS functionality as it is implemented in the SAS System. A full-length paper on this topic and example code is available at www.majaro.com.

Data Storage

Size of Data Sets

SAS data sets can be as large as you operating system will allow. Each operating system has limitations.

There is a maximum of 32,767 columns that may be defined in a data set, with a maximum row size of 1GB.

Under Windows, different formatted drives have different limits:

FAT16	2.1 gig
FAT32	4 gig
NTFS	>4 gig

MVS data sets may span multiple volumes.
Unix data sets have no size limitation.

Data set and column names may be up to 32. The maximum length of a character variable is 32K.

With the introduction of SPDE engine, extremely large data sets may now be partitioned for parallel I/O and processing. In this case, each logical data set is divided into multiple separate physical files called partitions. This engine also allows up to 2**31 columns in a data set.

Data Set Compression

Compressing large data sets may save physical storage space and I/O. The COMPRESS= data set option or system option is used to specify a compression algorithm. A row is automatically uncompressed when read.

COMPRESS=CHAR uses RLE (Run Length Encoding) and treats each row of a created data set as a single string of bytes, compressing multiple occurrences of the same byte without regard for column boundaries. This is best for rows with many character variables.

COMPRESS=BINARY uses RDC (Ross Data Compression), which combines RLE and sliding-window compression to compress rows, and is best for compressing numeric data.

Because of these compressing algorithms, compressing works best for data sets with a large record length. User-written compression algorithms may be also used.

Compressed data sets permit direct access of rows, just like uncompressed data sets.

Data Set Generations

Generation data sets are archived copies of a SAS data set that are automatically created whenever a data set is replaced. The copies have the same root data set name with different version numbers. The base version, which is the most recent version, plus the set of historical versions are referred to as a generation group.

The GENMAX= data set option specifies a number, up to 999, that indicates the number of data sets to maintain in the generation group.

There is no expiration on a generation data set. Data generations are only deleted when the maximum number of generations is exceeded.

When GENMAX is in effect, the data set name is limited to 28 characters, instead of 32. This is because the last 4 characters in the data set name are used to indicate the generation of the data set.

These last 4 characters of the SAS data set file name are #*nnn*, where *nnn* represents the historical version. The oldest version is 001, while the most recent version is *nnn*, the number specified in GENMAX.

When accessing the historical data, use the GENNUM= data set option to specify which generation to access. GENNUM has a default value of 0, the current version of the data set.

```
/* Print the oldest generation        */
proc print data=demog ( gennum = 1 ) ;
```

Since the *nnn* may not be known, using a relative index may be easier. These are negative numbers that are relative to the base version and reference the archived data sets from youngest to oldest.

```
/* Print the most recent generation  */
proc print data=demog ( gennum = -1 ) ;
```

Indexes

Index Files

An index is a separate file that can be created for a SAS data set to provide direct access to rows. The index stores values in ascending value order for a specific variable or variables and includes information as to the location of those values within rows in the data set.

Without an index, SAS accesses observations sequentially in the order in which they are stored in the data set. With an index, a data access is much quicker, but this comes at the expense of having to maintain the index.

SAS maintains the index as part of the data set, automatically updating the index as rows are added, deleted, or modified.

Multiple indexes may be defined for a data set.
Indexes may be defined on compressed data sets.

```
/* simple index named ptid        */
data demog (index=(ptid)) ;

/* compound index named x      */
data vitals (index=(x=(ptid visit)) ;
```

Indexes may also be defined via PROC DATASETS, SCL and SQL.

Unique Values

A unique index guarantees that values for the key variable(s) in the index remain unique for every observation in the data file. If an attempt to update the data set violates the index uniqueness, the update is rejected.

```
/* require PTID to be unique             */
data demog (index=(ptid)/unique)

/* require PTID VISIT to be unique    */
data vitals (index=(x=(ptid visit)/unique)
```

Missing Values

To keep variables that contain a large number of missing values from using space in the index use the NOMISS option to specify that missing values not be maintained by the index.

```
data demog (index=(ptid)/nomiss)
data vitals (index=(x=(ptid visit)/nomiss)
```

Missing values for the key variable can be added to the data file. The missing values are simply not added to the index.

Note that a NOMISS index cannot be used to process a BY statement or to process a WHERE expression. Because of this, it may be more desirable to create the index without the NOMISS option and then define a separate integrity constraint to keep missing data from being loaded into the data set.

Data Integrity

General Integrity Constraints

Integrity constraints allow you to define rules that a row must meet before it can be saved to a data set. For example, you can specify integrity constraints that requires the Patient ID value to be non-null and between 1 and 50.

Use of integrity constraints can assure the cleanliness of stored data. If an attempt to update a data set violates an integrity constraint, the update is rejected.

General constraints define rules for required values, unique values, acceptable values, or making the data values contingent on the value of another variable.

There are four types of general integrity constraints:

Check	Limits the data values in a variable to a specific set, range, or list. This constraint can also be used to make the data values in one variable contingent on the data values in another variable.
Not Null	Missing values for character and numeric data are not allowed. Numeric special missing values are also prohibited.
Unique	Requires that the specified variables contain unique data values. This creates a unique index and will use such an index if already defined.
Primary Key	Requires that the specified variables contain unique data values *and* that missing values are not allowed. This constraint creates a unique index. A data set can have only one primary key.

Integrity constraints cannot be created with data set options. PROC DATASETS, SQL, or SCL may be used to create integrity constraints. A simple PROC DATASETS to assign integrity constraints with custom error messages to the data set DEMOG is below:

```
proc datasets   lib=work ;
  modify demog  ;
   /*  Integrity constraint to require PTID  */
   ic create pt_req = not null (ptid)
      message = 'Patient ID is missing' ;
   /*  Integrity constraint for PTID range  */
   ic create ck = check(where=(0<ptid<51))
      message = 'ID not beween 1 and 50' ;
quit ;
```

Note that a not-null integrity constraint has a different effect than a NOMISS index. The integrity constraint prevents missing values in a SAS data set while the NOMISS index allows missing data values in the data set but excludes them from the index.

Referential Integrity Constraints

Referential constraints allow you to link the data values to variables in another data set. An example of a referential constraint would be linking the values for a Patient ID variable in a patient enrollment data set to a similar variable in an adverse events data set. Only the patients that exist in the enrollment data set would be allowed in the adverse events data set.

Referential integrity constraints, then, are comprised of two parts:

Primary Key	This part identifies unique rows in the lookup data set. This is the same as a general primary key integrity constraint.
Foreign Key	This part is referenced by values that are in the lookup data set's primary key values.

The referential integrity constraint also controls which actions will be taken when a primary key in the lookup table is deleted or modified:

Restrict	This prevents the primary key values from being deleted or modified when there is are any matching foreign key values. This is the default.
Set Null	Allows the primary key values to be deleted or modified. The foreign key values in the current generation of the data set are set to missing.
Cascade	Allows the primary key values to be modified. The foreign key values in the current generation of the data set are set to the new primary key values.

Referential integrity constraints cannot be assigned to data sets that maintain generations.

A PROC DATASETS to assign referential integrity constraints is illustrated below.

```
proc datasets lib=work ;
  /*  Create primary key constraint          */
  modify demog ;
    ic create pk = primary key(ptid) ;
  /*  Create foreign key constraint to       */
  /*  cascade changes to primary key         */
  /*  and null deleted primary keys          */
  modify ae ;
    ic create fk = foreign key(ptid)
       references work.demog
          on update cascade
          on delete set null
       message='Pt not enrolled.'  ;
quit ;
```

Primary key values are examined whenever a foreign key value is added or modified. This occurs even if the library containing the primary key is not currently allocated.

Similarly, foreign key values are examined and updated regardless of whether the library containing the foreign key is allocated. Because of this, the definition of many foreign keys that reference a single primary key table will involve substantial resources.

It is important to note that a data set that is involved in a referential integrity constraint cannot be renamed, deleted, or replaced. These actions require that the referential integrity constraint be removed from the data sets. The primary key cannot be deleted until all foreign keys that it references are first deleted:

```
proc datasets lib=work ;
  /*  Remove foreign key constraint  */
  modify ae ;
    ic delete fk ;
  /*  Remove primary key constraint  */
  modify demog ;
    ic delete pk ;
quit ;
```

The primary key and foreign key integrity constraints store data values in an index file. If an index file already exists, it is used. Otherwise, one is created.

When an index exists, the index's attributes must be compatible with the integrity constraint in order for the integrity constraint to be created. When adding a primary key constraint, the existing index must have the UNIQUE attribute. When adding a foreign key constraint, the index must *not* have the UNIQUE attribute.

Data Entry Integrity: Informats

Since an informat is an instruction for reading data, informats can be used to perform error checking as values are entered. Values resulting in the _error_ condition are flagged in error when input, but may be overridden.

```
proc format  library=keep ;
   invalue  $sex  (upcase just)
       'M'='M'      'MALE'='M'
       'F'='F'      'FEMALE'='F'
       other = _error_ ;
   invalue  $weight
       56 - 420 = _same_
       other    = _error_ ;
run ;
```

Data Entry Integrity: SCL

Regardless of whether data entry is performed via SAS/FSP or SAS/AF, the SAS Component Language (SCL) is available to code edit checks directly into the data entry screen.

Sample SCL for an Fsedit screen is shown below. To use a SAS/AF form viewer, change the errorOn/errorOff functions to calls to the _errorOnColumn/_errorOffColumn methods. Messages are displayed when HT is in out of range and an error that can be overridden is set on Sex when male is indicated to be pregnant.

```
test:     value:
   err = 0 ;
   select ( test ) ;
      when ( 'HT' )
         if not ( 44 < value < 91 ) then err=1;
   end ;
   if err then _msg_ =
     'Value for ' || test || ' is invalid.' ;
return ;

sex:      pregnant:
   dcl list msglist = { 'Verify pregnancy!' };
   if sex = 'M' and pregnant = 'Y ' then do ;
      button = messagebox (msglist , '!', 'O');
      errorOn sex ;
   end ;
   else  errorOff sex ;
return ;
```

Audit Trail

An audit trail logs modifications to a SAS data set in a separate audit data set. Each time a row is added, deleted, or updated, information is written to the audit trail identifying the who, what and when of the modification. The audit trail may be instructed to collect:

- Copies of a new row
- Copies of a deleted row
- Before and/or after copies of modified rows
- Failure to add, delete or update rows

The failure information that is tracked in an audit trail is not available anywhere else in the SAS System. These failures may be the result of insufficient authorization, for example, if another user has the row locked, or because a row is rejected because it does not meet a unique index or an integrity constraint.

The audit data set contains

- all variables in the data set
- any user-specified variables that appear only in the audit trail
- _at*_ variables with audit information.

  atDatetime Datetime of event
  atUserid Who made event
  atObsno Which row event
  atReturnCode Event return code
  atMessage Message in Log
  atOpCode Type of event

Values	Description
DA, DD	Data Add, Data Delete
DR, DW	Data Read, Data Write
EA, ED	Error Add, Error Delete
EU	Error Update

When data is rejected due to an integrity constraint, the custom message defined with the integrity constraint is loaded into the _atMessage_ field in the audit trail.

A simple example of using PROC DATASETS to initiate an audit trail is below:

```
proc datasets  lib=work ;
   /*  Initiate audit trail              */
   audit demog ;
      initiate ;
   /*  What goes in the audit trail  */
   log before_image =yes
       data_image    =yes
       error_image   =yes ;
   /*  User-specified variable       */
   user_var
       reason  $40  label='update reason' ;
quit ;
```

To access the audit trail, use the type=audit data set option:

```
proc print  data = demog ( type = audit ) ;
run ;
```

Do not use audit trails for data sets that will be moved, replaced or sorted in place. The audit trail cannot be maintained in such events and will be deleted. Because of this, SAS issues a warning in the Log if you initiate an audit trail for a data set that is not ALTER password protected.

If the data set is not part of a generation group, replacing the data set or sorting it with the FORCE option will cause the audit trail to be deleted without the possibility of recovery.

However, in cases where an audit trail is maintained for a generation data set, when the data set is replaced, a generation is maintained for both the data set and its audit trail. The audit trail for the newly created data set must be re-initiated. Both the TYPE=AUDIT and the GENNUM= data set options must be specified to access the audit trail generations.

In regards to clinical data, the default information maintained in a SAS audit trail is insufficient for CFR 21 Part 11 regulations from the FDA. This is because these regulations require that a 'reason for change' be maintained in an audit trail.

Requiring a 'reason for change' in the audit trail is easily accomplished in Version 8 of SAS. This is done by assigning a not-null integrity constraint to a user-defined 'reason for change' variable defined in an audit trail.

Recall that a user-defined variable for the audit trail appears in the audit trail only, not in the SAS data set. Since a user-defined audit trail variable is not part of the SAS data set, the ability to assign an integrity constraint on such a user-defined variable was deemed to be a bug by SAS. Integrity constraints are only for data set variables.

Thus, in Version 9 of SAS, this ability to assign an integrity constraint to a user-defined audit trail variable has been removed. This seems a curious removal of functionality that now requires developers in our industry to devise alternate methods to meet the 'reason for change' requirement in an audit trail for CRF 21 Part 11 compliance.

Data Security

Data Set Password Protection

There are three levels of password protection that may be defined when a data set is created:

READ=xxx	Restrict read access.
WRITE=yyy	Restrict ability to add, delete, or modify rows.
ALTER=zzz	Restrict ability to modify data set structure or replace the data set or index.

Passwords must be 8 characters or less and are not case sensitive. Passwords are the same on all data sets in a generation group as well as the audit trail.

These various levels of security provide the various levels of user permissions necessary in a clinical DBMS environment. Read access is available to data reviewers, write access for data entry and cleaning, alter access for DBAs.

Do not lose your passwords. Only by using a lengthy and difficult process is the SAS Institute able to recover data when passwords are lost.

Data Set Encryption

Data sets may be encrypted when they are created by using the ENCRYPT=YES data set option. A READ= password must be specified since the encryption algorithm uses this password. Because of this, you cannot change a password on an encrypted data set without re-creating the data set.

SAS data set encryption is based on a 31-bit key and is characterized as "moderately effective" or "medium grade" security.

If a data set is encrypted, all associated files, such as indexes, the audit trail and data generations, are also encrypted. Encryption requires roughly the same amount of CPU resources as data set compression.

Data Access

Data Views

A SAS view does not contain any data values. Instead, views contain references to data that are stored elsewhere. Views can be used to avoid storing data elements in numerous locations or to define specific subsets of data. Views also ensure that accessed data is always current because data from views are derived at run time.

While a SAS data view may be password protected, you cannot encrypt, index, or audit views because they contain no data.

The following examples demonstrate how to define a view using a Data Step and SQL:

```
data lib.b_view  / view = lib.b_view ;
   infile 'k:\source_file.txt' ;
   input  ptid 8   item $  value 8 ;
run ;

proc sql  ;
   create  view  lib.a_view (read = pw ) as
      select  *
      from n_drive.demog ( read = fff ) ,
           x_drive.demog (read = ggg)
run ;
```

Another important function that a view can serve is the elimination of I/O bound processing by avoiding unnecessary temporary tables. Printing the view created above, lib.a_view, involves much less I/O than creating an intermediate data set that contains all the data defined by the view and then printing the created data set.

Query Optimization

Query optimization is another important aspect of a DBMS. Different DBMS systems handle queries differently. The SAS System performs an algebraic manipulation of WHERE expressions to place the clause into conjunctive normal form to simplify expressions and take advantage of indexes that may exist.

Indexes can be considered intelligent because they contain distribution statistics for the indexed values. By examining the index distribution, SAS is able to optimize a WHERE expression by estimating the number or rows that the WHERE will return and determine whether to use the index or a sequential read. Usually if the index will return less than %30 of the table, it will be used to read the data.

Although views cannot be indexed, if the view definition includes a WHERE expression using a key variable, then the index will be examined for usage. Accessing the view with a WHERE will push the WHERE expression to the view and evaluate the entire WHERE expression.

The SQL query optimizer employed by SAS performs additional optimizations such as

- Data Flow Analysis
  Unnecessary variables are removed from various stages of processing to reduce the size of intermediate results.
- Join strategies
  Evaluate expressions that apply to a single table, then determine whether to implement joins by using indexes, sorting, or a hashing.
- Subquery strategies
  Subqueries are evaluated once and results are cached for quick lookup.
- Set Operator strategies
  The SQL set operators, Union, Intersect, Except, are typically more costly than corresponding SAS Data Step processing.
- Implicit Pass Thru
  When accessing non-SAS DBMS, SAS will convert the appropriate parts of the SQL to the DBMS-specific SQL and pass the converted SQL to the DBMS for evaluation.

For further details, see the chapter on SAS Indexes in the SAS OnlineDoc and SAS TS-320, "Inside PROC SQL's Query Optimizer", at sas.com/techsup/download/technote/ts320.htm.

Row-Level Locking
This important feature, provided by SAS/SHARE, permits concurrent access to data in a table.

A separate SAS session on a server machine is used to establish a SAS/SHARE server. This session only executes a special procedure, PROC SERVER, which listens for and acts upon requests from client SAS sessions to serve and write data.

Any number of client SAS sessions may then use the SAS/SHARE server after it is initialized. This only requires that the server be identified on the libname statement.

When a user locks a row, for example by displaying the row in Fsedit, other users may view the row in browse mode only. Although many users may have the table open in write mode, only one user may lock a row at any given moment.

SAS/SHARE*NET is a separate product that acts to manage requests from client sessions in much the same fashion as SAS/SHARE. The difference is that this product manages requests from non-SAS clients, such as web pages.

Performance and Scalability
Version 9 of the SAS System has greatly enhanced abilities for parallel I/O and parallel computing that can take advantage of hardware capabilities and large memory configurations.

Multi-threaded I/O requires multiple I/O channels and multiple disk drives. Multi-threaded processing requires multiple, single processor machines on a network, a symmetric multi-processing (SMP) machine, or a combination of both.

The various techniques for parallel processing can be summarized as follows:

- Threaded Kernel
  The Threaded Kernel in Base SAS allows certain CPU intensive procedures, such as SORT, SUMMARY, GLM and REG to now execute in parallel threads on multiple CPUs. SQL processing is also done in parallel with other Base processing. The system option NOTHREADS can be set to suppress the use of threads.
- MP Connect
  Multi-process Connect is part of the SAS/CONNECT product and permits parallel processing in multiple remote SAS sessions. Implementing this functionality requires SAS/CONNECT syntax changes to an existing application.
- SPDE Engine
  The Scalable Performance Data Engine, available in Base SAS, is for use with massive data sets and provides capabilities for both parallel I/O and processing. A data set and its indexes must be partitioned when created in order to permit this engine to process each partition in parallel. Special options on the libname statement are required. This engine also takes advantage of very large memory configurations by keeping as much data as possible in memory.
- I/O Engines
  The Base SAS I/O engine, many SAS/ACCESS engines as well as the SPDE engine all now read blocks of data, as opposed to one row at a time. Additionally, some SAS/ACCESS engines support bulk loading and multi-row writes.
- SPD Server
  The Scalable Performance Data Server, a separate SAS product, provides all the features of the SPDE engine and additional functionality such as userid/password validation, backup and recovery facilities, catalog and catalog entry protections, and table, table column, and table row protections.
- Open Metadata Server
  Provides parallel serving of data from common metadata repositories used by various SAS Solutions.
- OLAP Server
  Parallel I/O and processing of multidimensional cubes of data. Requires changes to the server configuration file and may be controlled from the client.

Conclusion

As you can see, the SAS System certainly provides the functionality of any DBMS.

SAS is also a simpler solution to implement than other DBMS. I found this interesting quote by Brooks Tally in InfoWorld: “In general, Oracle products are designed for very professional development efforts by top-notch programmers and project leaders. The learning period is fairly long, and the solution is pricey”.

This is quite a statement. Our industry may be experiencing an irrational exuberance in embracing an inappropriate solution. A DBMS that is optimized for On-Line Transaction Process (OLTP), such as customer service and stock exchange systems, is not the appropriate solution for the On-Line Analytical Process (OLAP) needs of a clinical DBMS.

SAS is the industry leader in providing OLAP data warehousing optimization. After this review, I argue that SAS is *the* DBMS that is appropriate for the Clinical Trials industry.

References

Doninger, Cheryl (2002), “Up and Out: Where We’re Going with Scalability in SAS Version 9”, *Proceedings of the Twenty-Seventh Annual SAS Users Group International Conference*

Franklin, Gary (2000), “Integrity Constraints and Audit Trails Working Together”, *Proceedings of the Twenty-Fifth Annual SAS Users Group International Conference*

Rhodes, Diane Louise (2001), “Migrate to Oracle? I need my SAS!”, *Proceedings of the Twenty-Sixth Annual SAS Users Group International Conference*

About the Author

Steve Wilson is employed as the Director of Clinical Applications Development at MAJARO InfoSystems, Inc. While at MAJARO, Steve’s primary responsibility has been development of ClinAccess 5.0 ™, a clinical trials software package written in Version 9 of SAS the SAS System. Steve has been developing applications in SAS for nearly 20 years and is a Version 8 SAS Certified Professional. He has given numerous presentations at SUGI as well as regional and local conferences.

Steve can be contacted at: SWilson@majaro.com