Paper 112-29

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 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.

At the time of this writing, the FDA has not issued revised CFR 21 Part 11 guidance. This indicates that the FDA is still struggling with the difficult issue of Part 11 requirements compliance for a clinical database. However, it is known that the FDA does request that all clinical DBMS prevent unauthorized access to data. This requires that data be password protected and allow updates only by authorized users of the clinical DBMS.

The FDA also requires maintenance of complete audit trails for clinical data. Both the entry and revision of clinical data is to 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 and many others are all easy goals when working with the SAS System. In addition to being easy to use, SAS is also the choice of the FDA for receiving and reviewing clinical data. The FDA is still requesting SAS transport data sets as the standard for receiving electronically submitted data. At a minimum, each FDA reviewer is equipped with the SAS System Viewer.

Since the use of SAS is essential to clinical information management, keeping clinical data in SAS makes sense.
SAS is the government and the industry standard for performing clinical analysis. 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? It is certainly a file of raw information, as any text miner will attest.

Frequently, the term database is used when actually referring to a Database Management System (DBMS) – a database with advanced functionality. This DBMS functionality should assist with speedy data management and ensure the security and audit-ability of the data. The actual database itself 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

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 for not using SAS as a clinical DBMS 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 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. Since SAS is already running in 90% of the Fortune 500, why maintain a separate DBMS for clinical trials data?

The management and analysis of clinical trials data is best performed in a DBMS that is tuned for On-Line Analytical Processing (OLAP). Many popular DBMS products are tuned for On-Line Transaction Processing (OLTP), which is necessary for many hits on a massive data structure requiring immediate response. Clearly, clinical trials data management does not require an OLTP solution. This fact clearly illustrates why SAS is a better Clinical DBMS solution than an OLTP DBMS like Oracle.

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 examines each of the previously listed functional expectations of a DBMS and how the functionality is implemented in SAS. At the end of this review, I hope to have you agree that SAS is indeed a DBMS solution that is appropriate for the management of clinical trials data.

SAS DATABASE FUNCTIONALITY

DATA SET OPTIONS AND PROC DATASETS

Many database features in SAS are assigned via data set options or via PROC DATASETS. Data set options are numerous and control certain attributes of a data set. PROC DATASETS is used to manage data sets and performs various actions to a data set. It is important to learn the functionality provided by each.

SAS SOFTWARE PRODUCTS

Your specific goals will determine the blend of SAS System products you will need for your solution. Your budget may determine which SAS System products are available. The following table gives a quick overview of the SAS System products most relevant to a clinical DBMS.

Basic database management	Base SAS
Concurrent access to data.	SAS/SHARE
Allow the SAS/SHARE server to manage requests from a non-SAS client application (eg. Web page)	SAS/SHARE*NET
Submission of SAS code to a remote host and permit parallel processing.	SAS/CONNECT
Strong network encryption.	SAS/SECURE
Custom built user interfaces: Client/server application.	SAS/AF
Custom built user interfaces: Thin-client web-based application.	AppDev Studio SAS/WebAF SAS/IntrNet
User interface for managing a Data Warehouse	SAS/Warehouse Adminisrator
Engines for non-SAS files	SAS/ACCESS
Data and text mining	Enterprise Miner

DATA STORAGE

SIZE OF DATA SETS

SAS data sets can be as large as your 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.
The SPDE engine allows extremely large data sets to 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 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 or COMRESS=YES 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. User-written compression algorithms may also be used.

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

Both of these compressing algorithms work best on data sets with a large record length. Since this is a data dependent function, testing within your application is necessary to determine which is best. Generally the RDC will give better compression when the row length is greater than 1000 bytes,

When the COMPRESS= data set option is used, the REUSE= option is also available for use. This option is a Yes/No indicator as to whether new observations to the compressed data set are appended to the end of the data set or whether the new observation is inserted into space freed by deleting or updating other observations. By specifying REUSE=YES, you are able to achieve more efficient data storage with compressed data sets that have many update or delete operations.

Compressed data sets permit direct access of rows. This is accomplished by automatically embedding a special system index into the compressed data set. This index is not created when using the REUSE=YES data set option to reuse space from deleted rows. The REUSE=YES option is incompatible with this system index, so direct access of rows is only possible in this situation by using an explicitly defined index.

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. Example host file names under Windows are dsname.sas7bdat and dsname#001.sas7bdat.

To access 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.

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

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.

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

INDEXES

INDEX FILES

An index is a separate file (dsname.sas7bndx under Windows) that can be created for a SAS data set to provide nearly instantaneous direct access to rows in the table. The index stores values in ascending order for the specified index variable or variables and a pointer to the location of rows with those values within 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, automatically updating the index as rows are added, deleted, or modified in the data set.

Multiple indexes may be defined for a data set. Indexes may be defined on compressed data sets.
The following code illustrates how to create indexes using a data set option. Indexes may also be defined via PROC DATASETS, SCL and SQL.

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

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

INDEX: 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.

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

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

INDEX: 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=(other)/nomiss);
   data vitals (index=(x=(ecg qt)/nomiss);
```

Missing values for the key variable can be added to the data file. The missing values are simply not added to the index. It is important to note that a NOMISS index cannot be used to process a BY statement or to process a WHERE expression. Thus, much functionality of an index is removed when the NOMISS option is used.

DATA INTEGRITY

GENERAL INTEGIRTY CONSTRAINTS

Use of integrity constraints can assure the cleanliness of stored data. 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. If an attempt to update a data set violates an integrity constraint, the update is rejected. Data that violates an integrity constraint can never be saved into a data set.

There are four types of general integrity constraints. General constraints define rules for required values, unique values, acceptable values, or making the data values contingent on the value of another variable.

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. 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.
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. Instead, 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  ;
   ic create                         /* Require PTID  */
     pt_req = not null (ptid)
       message =
         'Patient ID is missing'    ;

   ic create                         /* PTID range  */
     ck = check(where=(0<ptid<51))
       message = 'ID not in range'    ;
quit ;
```

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 variable of the same name in an adverse events data set. Patient IDs entered in the adverse events data set are restricted to the Patient IDs that exist in the enrollment data.

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 references the 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 in the lookup data set when there is are any matching foreign key values. This is the default.
Set Null	Allows the primary key values to be deleted in the lookup data set. The foreign key values in 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 ;
  /* Primary key constraint  */
  modify demog ;
    ic create
      pk = primary key(ptid)   ;

  /* Foreign key constraint to cascade changes to primary key  */
  /* and set to null any 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, defining many foreign keys on a primary key table will involve substantial resources.

If the changes to a primary key result in a foreign key value that violates a different integrity constraint, the update will not be permitted.

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 ;
     modify ae ;
        ic delete fk ;              /*  Remove foreign key   */
     modify demog ;
        ic delete pk ;              /*  Remove primary key   */
   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

Informats can be used to perform error checking as values are entered. Values resulting in the _error_ condition are flagged in error and cannot be saved to the data set. Unlike with an Integrity Constraint, this error condition may be overridden via the OVERRIDE command during data entry.

```
   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.

SCL is a powerful and mature language that is separate from Base SAS. Although it is a vast language, it is very easy to get started using SCL for simple error checking.

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 and use the code as the Model SCL.

The first section of code displays a message when HT is out of range. The second section sets an error on Sex when male is indicated to be pregnant. This error can also be overridden via OVERRIDE.

```
test:
value:
  err = 0 ;
  select ( test ) ;
    when ( 'HT' )
      if not (44 <value <91) then err=1 ;
    when ( 'WT' )
      if not (79 <value <401) then err=1 ;
   end ;
   if err then _msg_ =
     'Invalid ' || test || ' value.' ;
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 of data in a SAS data set to 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
- optional 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

The _atOpCode_ variable uses these values to indicate the action logged by the audit trail record:

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. The default SAS System error message is used if there is no custom message.

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

```
proc datasets  lib=work ;
    audit demog ;
        initiate ;                 /* Initiate audit trail       */

    log before_image  = yes        /* What goes into audit trail */
          data_image  = yes
          error_image = yes ;

   user_var                        /* User-specified variable    */
    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 ;
```

The audit trail file is actually not a SAS data set, but it may be treated like a SAS data set when it is correctly referenced using the TYPE=AUDIT data set option. Under Windows, the audit trail file name would appear as dsname.sas7baud.

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 automatically deleted. Because of this, SAS will issue 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 by SAS.

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 FDA regulations. This is because these regulations require that a 'reason for change' be maintained in an audit trail.

It may seem like a good idea to place a Not Null integrity constraint on the user-defined audit trail variable. By requiring a 'reason for change' value in the audit trail, FDA regulations regarding this requirement would be satisfied. DO NOT DO THIS. While this is possible in Version 8 of the SAS System, this functionality has been removed in Version 9 of SAS. Integrity constraints are only for data set variables only.

Thus, in Version 9 of SAS, you cannot assign an integrity constraint to a user-defined audit trail variable. 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 FDA compliance. I would urge the SAS Institute to reconsider the removal of this important functionality.

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.

These 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. 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.

While these passwords provide protections while within a SAS session, they do not prevent viewing the files at the operating environment level or from being read by a non-SAS process. Operating system permissions must be used in conjunction with the SAS passwords to provide comprehensive security.

Do not lose your passwords. Only via 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. SAS data set encryption is based on a 31-bit key and is characterized as “moderately effective” or “medium grade” security. While SAS data sets are stored in a proprietary structure, the SAS data set structure is an open interface to Microsoft ODBC and Java JDBC, which is why you may wish to encrypt SAS data sets.

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 and SAS automatically decrypts the data as it is read.

When creating an encrypted SAS data set, 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.

NETWORK ENCRYPTION

Network encryption is provided with the SAS/SHARE and SAS/CONNECT products. Network traffic is generated when a SAS session on a server and a SAS session on a client are communicating. In SAS/CONNECT terminology, these are the remote and local SAS sessions respectively. This feature will encrypt data before it is passed off as network traffic. The data is decrypted as it is received by either the client SAS session or the server SAS session.

SAS Version 9 makes available the Secure Sockets Layer (SSL) network security protocol for strong encryption. Strong encryption prior to V9 is available via the SAS/SECURE product. There is not a comparable strong encryption for available SAS data sets.

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 SQL:

```
proc sql  ;
  create view lib.view2 (read=pw) as
    select  *
    from n_drive.demog (read=pw_n),
         x_drive.demog (read=pw_x)
    where ptid gt 150 ;
run ;
```

This example illustrates how a view can serve to elimination of I/O processing by avoiding unnecessary temporary tables. Printing the view created above, lib.view2, involves much less I/O and storage than creating an intermediate data set that contains the data defined by the view and then printing the created data set.

It is important to note that n_drive and x_drive must already be allocated when the view lib.view2 is referenced. Unlike Integrity Constraints, SAS views do not automatically access libraries that are part of the view definition.

This next example demonstrates how to define a view using a Data Step. This example illustrates how to create the illusion that an external file exists as a SAS data set:

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

The view lib.view, which is used like a regular SAS data set, reads the ASCII file and processes the WHERE before passing the requested data to the PROC PRINT.

```
proc print data=lib.view
  where ptid gt 150 ;
run ;
```

Because the GENNUM= data set option cannot be used with a view, when creating a view of a generation data set, the view must explicitly reference the desired generation. Therefore, when creating a view of a generation data group, a separate view of each generation must be explicitly defined. For example:

```
proc sql  ;
  create view first_demog as
    select *
    from a.demog (gennum=1)  ;
```

QUERY TOOLS

The SAS System provides an excellent interface for programmers and end users alike. Syntax is easy to learn and interactive interfaces exist for performing vast amounts of data manipulation and management.

Custom GUI client/server applications can be developed with SAS/AF and the AppDev Studio products. SAS/AF develops fat-client applications that require a SAS session to be running on the client, which distributes more processing. The AppDev Studio on the other hand, creates thin-client web-based applications, where only the server is running a SAS session.

Reports and graphics can be OLAP enabled, allowing point-and-click data manipulations and drill-down capabilities. Structured Query Language (SQL) is implemented in SAS via PROC SQL. SQL is a standardized, widely used language that retrieves and updates data in sets and views. A fully functional DBMS will process SQL queries efficiently.

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.

Consider the following code:

```
proc print data=lib.view2
  where sex = 'F' ;
run ;
```

Only data meeting the condition WHERE PTID GT 150 AND SEX='F' are read from the n_drive.demog and x_drive.demog data sets and passed to the PROC PRINT. There is an I/O savings realized because a temporary data set is not created to contain the desired data. Additional I/O reduction is achieved when there are indexes on the source data sets that are able to directly access the desired data.

The SQL query optimizer employed by SAS performs additional optimizations. Among these optimizations are the following:

- 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 DBMS feature, provided by SAS/SHARE, permits concurrent access to data in a table. This allows multiple users to edit the same data set at the same time.

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 use the SAS/SHARE server after it is initialized. This only requires that the server be identified on the libname statement in the client SAS session.

When a user locks a row, for example by displaying the row in Fsedit, other users may view the row in browse mode only. Many users may have the table open in write mode, but only one user may lock a row at any given moment. Attempts to lock a row that is already locked by another process will send a message to the secondary attempt indicating that the row is locked by another process.

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 automatically 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 multiple rows 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. Requires changes to the server configuration file.
- 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.

RELIABILITY

The SAS System detects when any data set is in error. In the unlikely event that a data set is not closed properly or an index or integrity constraint is in error, SAS will automatically take appropriate actions to repair errors.

Index files are automatically re-created if they cannot be located. This may happen if the index file is deleted or when the data set is moved or renamed, but the index file is not.

In some cases, SAS may prompt for the Alter password of the data set when re-building files.

A deleted or damaged audit trail file will prevent accessing the data set until the audit trail is terminated and then initiated via PROC DATASETS. Be warned that in these cases the messages in the SAS Log offer no clue that the problem is due to the data set audit trail.

In extreme circumstances, the REPAIR statement of PROC DATASETS may be necessary to repair truncated data sets.

When SAS Views are damaged, the view must be re-created.

DATA INTEGRATION, WAREHOUSING AND MINING

The Warehouse Administrator manages the process of distributed data handling and content control by enabling access to legacy and non-legacy data sources in a highly flexible, easily maintainable environment.

To access other file structures and DBMS data, the SAS System provides different data engines that permit reading and writing to various other data structures, such as Oracle, DB2, ODBC, and SAP. Most of these engines require separate SAS licenses for the SAS/ACCESS engine. Once the external file is allocated with a libname statement, the data in the external file may be processed like a regular SAS data set.

In the other direction, there are open interfaces to SAS data tables via Microsoft ODBC and Java JDBC. SAS transport files are easily created and are also an open specification, described at www.sas.com/techsup/download/technote/ts140.html.

Data mining functionality, including text mining, via Enterprise Miner is only the newest addition to a long tradition of SAS providing the best tools available for analysis. According to AMR Research, in 2001 SAS held 21% of the market share in business intelligence and analytics. In surveys of products used for data warehousing, analytic applications, business intelligence, and CRM, SAS is the only software company to be ranked either 1st or 2nd every year the survey has been performed since 1998 by DM Review.

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”.

He continues, “If your project has tight deadlines and you don’t have the time and/or money to hire a team of very expensive, very experienced developers, you may find that the Oracle solution is an easy way to get yourself in trouble”. This is quite a statement. Our industry may be experiencing an irrational exuberance in embracing an inappropriate solution.

A DBMS like Oracle that is optimized for On-Line Transaction Process (OLTP) to support functions such as customer service or reservations systems is not the best solution for the On-Line Analytical Process (OLAP) needs of a clinical DBMS.

SAS is the industry leader in providing OLAP data warehousing optimization, analysis and reporting.
After this review, I would challenge the reader to demonstrate that SAS is not a DBMS. Indeed, I assert 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*

Levine, Fred (2001), “Using the SAS/ACCESS Libname Technology to Get Improvements in Performance and Optimizations in SAS/SQL Queries”, *Proceedings of the Twenty-Sixth 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

APPLICATION EXAMPLE

Sample application code that demonstrates the creation of a secure, auditable database is available at Steve Wilson’s Expert Corner at www.majaro.com.

This example also expands upon the techniques illustrated in this paper by demonstrating various methods to enter, validate, and update data sets that utilize the DBMS functionality previously discussed.

CONTACT INFORMATION

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