Efficient Use of the SAS/ACCESS Interfaces to INGRES® and SYBASE®

Jana Van Wyk, SAS Institute Inc., Cary, NC

ABSTRACT

Writing efficient SAS® applications using the SAS/ACCESS® interface to INGRES® or SYBASE® is easier if the SAS applications developer is aware of the internal techniques and algorithms used by the database interface. This paper discusses design strategies and performance considerations for users who will develop applications using the SAS/ACCESS interfaces to INGRES and SYBASE relational database management systems.

Included in the discussion of the architecture of the SAS/ACCESS interfaces is the ACCESS procedure, the DBLOAD procedure, the database I/O engine and the SOL, FSEDIT and FSVIEW procedures.

This paper is intended for users who will be developing SAS applications using the SAS/ACCESS interfaces to INGRES or SYBASE. This paper is not intended as a tutorial and will therefore assume some working knowledge of the SAS/ACCESS interfaces and the INGRES or SYBASE relational database management systems.

INTRODUCTION

INGRES and SYBASE are relational database management systems (DBMS) that run on a variety of hardware platforms. The Institute's goal is to make the SAS/ACCESS interface to INGRES running under the VMS environment available with Release 6.07. The SAS/ACCESS interface to SYBASE is currently under development.

Writing efficient SAS applications using the SAS/ACCESS interfaces to INGRES or SYBASE is easier if the SAS application developer is aware of the underlying techniques used by the database interface. This paper discusses design strategies and performance considerations for users who will develop applications using the SAS/ACCESS interfaces to INGRES and SYBASE.

This paper will describe the internal structure of the interfaces to INGRES and SYBASE and the architecture of all relational SAS/ACCESS interfaces including:

• the ACCESS procedure, which is used to create and edit SAS files called access and view descriptors (descriptors describe the data in the database)
• the database I/O engine, which is used by the DATA step and procedures to read, add, delete and update the data in the database based on the contents of view descriptors
• the DBLOAD procedure, which creates and loads a database table with data from a SAS data set, a view descriptor of a database table or a SAS SQL view.

The main goal of sharing the information about SAS internals in this paper is to allow the user of the SAS/ACCESS interfaces to develop better applications by avoiding the dangers that are inherent in transparent data access. Users will learn:

• when the SAS/ACCESS interface accesses the database
• how the software makes calls to the database via the routines provided by the database vendor
• how to create access and view descriptors for efficient use
• where the issues of possible performance problems occur
• how to anticipate and avoid these performance issues.

There are three sections in this paper that address these issues. They are:

• Overview of the SAS/ACCESS Architecture
• The DBMS Interfaces to INGRES and SYBASE
• Performance Considerations

This paper is not introductory; therefore, I will assume basic knowledge about the SAS/ACCESS interfaces and descriptors. Any details discussed in this paper will refer to the relational database interfaces to INGRES and SYBASE only, although some of the general comments are true of all the SAS/ACCESS interfaces.

There are several papers in the Proceedings of the Fifteenth Annual SAS Users Group International Conference which are tutorial and provide an overview of the SAS/ACCESS interfaces. The REFERENCES section at the end of this paper lists several sources for more information.

OVERVIEW OF SAS/ACCESS ARCHITECTURE

Discussion of the architecture of the SAS/ACCESS interfaces involves references to portable code, host code, database-specific code and the routines supplied by the database vendor that can be called from the database-specific code. Portable code is SAS code written in C which is portable across all hardware platforms. Host code runs on only one host machine. Database-specific code is C code that is not tied to an operating system but is tied to the specific internals of a database management system, such as INGRES.

The fourth type of code is the callable run-time interface that is a set of routines provided by the vendor of the DBMS. The database-specific SAS code makes calls to this run-time interface to communicate with the DBMS.

The ACCESS Procedure

The power of the SAS/ACCESS database interfaces comes primarily from using the database engine's ability to read and update the DBMS data directly from SAS programs without making a copy of the data and storing it in a SAS data set. However, the database engine cannot be used without having knowledge about which tables and columns are to be read from which database. The ACCESS procedure solves this problem by capturing all of the information needed to access a table or view in a database and storing this information in SAS external files called access and view descriptors.

Access descriptors are created to describe the contents of a DBMS table or view including details such as database name, table name, column names, the corresponding SAS variable names and the SAS formats and informats that will be used to convert the data. View descriptors are created from access descriptors, and each view descriptor can define all or any subset of the columns described in the access descriptor. A subsetting restriction, which is an SQL clause that limits the number of rows retrieved, can be added to any view descriptor.

There are two main differences in the contents of the descriptors for INGRES and SYBASE: the information needed to connect to the database and the database data types that describe the contents of the table. For the SAS/ACCESS Interface to INGRES, there are two pieces of database-specific information that must be entered when creating an access descriptor: the database name and the table name. For SYBASE, five fields can be entered in the access descriptor: the user's name, the user's password, the server name,
the database name and the table name. SAS software honors the normal SYBASE default action, so if the user name is not specified then the userid of the current process will be used; if a password is not given then NULL will be assumed; if a server name is not given then the server name will default to SYBASE; and if a database name is not given then the database name will be the default database name for the user. The INGRES database engine can connect to more than one database in one SAS step, and the SYBASE engine can connect to more than one database and more than one server in one SAS step.

Figure 1 shows the modularity of the tasks of the ACCESS procedure. Note that circles refer to physical files and boxes refer to executable code.

![Figure 1: Modularity of the tasks of the ACCESS procedure](image)

After the ACCESS procedure is invoked, code that controls the screens is executed. If descriptors are created or edited, another section of code controls this I/O. The code is modularized so that the database-specific code that communicates with the database is called only once when creating an access descriptor. This minimizes the impact on the database. The calls are made to the DBMS only when an access descriptor is created; when a view descriptor is created. information is read from an access descriptor.

The majority of ACCESS procedure code controls the reading and writing of the user interface window screens and the reading and writing of the descriptor files. The database-specific code must fill in the contents of the structures within the descriptors, make queries to the database regarding the contents of a certain table in a database and translate the database data types into corresponding SAS informats.

At this time, the ACCESS procedure allows only a full screen interactive interface for the user. However, support for SAS line mode syntax for the ACCESS procedure is under consideration for the INGRES and SYBASE interfaces. SAS line mode syntax can be used in interactive or batch processing.

### Database I/O Engine

Figure 2 illustrates the relationship between the SASIACCESS software and the DBMS when the database I/O engine is used by the PRINT procedure.

![Figure 2: Database I/O Engine](image)

The job of the database engine is to accept any request from the SAS supervisor, translate it into the appropriate DBMS request and return data or status information to the SAS supervisor in the correct SAS format. For example, if a procedure asks the SAS supervisor to read a record, the database engine will build a DBMS read request, pass it to the database system and receive the data that was read. Then the engine must translate the DBMS data into the format that the engine supervisor expects and pass the translated data to the SAS supervisor, who passes it on to the procedure.

Let's take a closer look at the database I/O engine functions that occur when this SQL code is executed:

1. The SAS supervisor reads the engine header record in the view descriptor and loads the appropriate database I/O engine.
2. The engine reads the view descriptor to glean database connection information and row subsetting information.
3. The engine connects to the database.
4. The engine builds the SQL select statement:
   ```sql
   SELECT EMPLOYEENNAME, EMPLOYEEID
   FROM EMPLOYEES
   WHERE EMPLOYEEID > 100
   ```
5. and then appends the subsetting restriction:
6. The engine fetches the DBMS data from the database.
7. The engine translates the data into SAS types using the formats in the view descriptor.
8. The engine passes back this information to the engine supervisor, recording any errors.

The database engine allows both read-only and update access to the database table. Virtually any procedure in the SAS System can be used to read the DBMS data via the database I/O engine. The engine can update the contents of a DBMS table when used by the FSEDIT procedure, the FSVIEW procedure, the APPEND procedure or the SAL procedure.

### The DBLOAD Procedure

Figure 3 illustrates the relationship between the SASIACCESS interface and the DBMS when the DBLOAD procedure is executed. After the PROC DBLOAD statement is typed, the code that parses the SAS line mode commands is executed, then the
The DBLOAD procedure creates and loads a table and provides a mechanism to allow the user's ad hoc SQL statements (with the exception of a SELECT statement) to be sent to the database. The procedure can load a table using data from a SAS data set, an SQL data view or data described by a view descriptor of another DBMS. Part of the philosophy behind the architecture of the SAS/ACCESS interfaces to INGRES and SYBASE is that, in almost every case, they will not override the DBMS default handling of these situations: inserting, locking, data integrity, security, journaling, system interrupts and transaction processing. They are handled by the DBMS run-time interface and not by the SAS/ACCESS interfaces' database-specific code. This guarantees that if a user is accustomed to doing data processing directly with a DBMS, when they use the SAS System to access data in a database, the behavior they expect from the DBMS will be honored by the SAS System. There are cases in which it is desirable for the SAS System or the user to override these defaults and this will be discussed later in Effective Use of Engine Locking and Transaction Processing.

THE DBMS DEFAULTS

Part of the philosophy behind the architecture of the SAS/ACCESS interfaces is that, in almost every case, they will not override the DBMS default handling of these situations: inserting, locking, data integrity, security, journaling, system interrupts and transaction processing. They are handled by the DBMS run-time interface and not by the SAS/ACCESS interfaces' database-specific code. This guarantees that if a user is accustomed to doing data processing directly with a DBMS, when they use the SAS System to access data in a database, the behavior they expect from the DBMS will be honored by the SAS System. There are cases in which it is desirable for the SAS System or the user to override these defaults and this will be discussed later in Effective Use of Engine Locking and Transaction Processing.

THE DBMS INTERFACES TO INGRES AND SYBASE

Interface to INGRES - dynamic INGRES/Embedded SQL™

The run-time interface used by the SAS System to access data in an INGRES database is called dynamic INGRES/Embedded SQL. SQL statements are formed dynamically as strings of text and passed to the INGRES/Embedded SQL routines to be executed. Control structures named cursors are used to read and write data and an SQL pre-compiler is used precompile code that contains embedded SQL statements. For example, when using a SAS view descriptor to read the contents of the table EMPLOYEES, the INGRES database engine can load a table using a SAS WHERE statement with an INGRES database engine.

1. connects to the database using connection information from the descriptor.
2. forms the SQL statement as an ASCII string, for example,
   ```sql
   SELECT EMPID FROM EMPLOYEES
   ```
3. prepares the SQL statement. At this point syntax and semantic checking is performed on the SQL statement, so errors will be caught before execution.
4. describes the SQL statement. The DBMS routine fills in a structure detailing the information needed to process the data retrieval.
5. opens a cursor so that data retrieved can be read.
6. fetches each row of data from the cursor and converts the data into a SAS format, if necessary.
7. closes the cursor, signaling the end of the data retrieval.
8. commits this database transaction, freeing all locks placed on pages of data.
9. disconnects from the database, if there are no other active SAS tasks that are connected to this database.
10. Several other SAS/ACCESS interfaces were implemented by using dynamic embedded SQL, including the DB2™, SQUIDS™, Database Manager™ and RdbVMS™ interfaces.

Interface to SYBASE - DB-Library™

The SYBASE client/server architecture consists of two components: the SQL Server™ and the SQL Toolset™. The database interface used by the SAS System will be part of front-end, or client, toolset application code that is separated from the back-end database server functionality. The database engine uses a programming interface named DB-Library, which is client software that communicates with the SQL Server. The engine calls these routines that send Transact-SQL commands to the SQL Server and process the results of these commands. Other routines handle error conditions, perform data conversion, and provide a variety of information about the interaction with the SQL Server. The DB-Library approach is distinctly different from the dynamic embedded SQL interface that INGRES provides. In contrast to an embedded SQL interface, the DB-Library interface does not require a host language precompiler and does not make use of cursors when SELECTing data.

Programming with DB-Library typically involves a few basic steps. Here is an example of how the SYBASE database engine executes when a view descriptor is used in a PRINT procedure statement. The engine
1. logs into the SQL Server using the server name, user name and password information in the view descriptor.
2. places the Transact-SQL command SELECT EMPID FROM EMPLOYEES into a buffer and sends them to the SQL Server, which may be installed on another node in the network.
3. processes the results, either errors or data, returned from the server.
4. closes the connection with the SQL Server.

PERFORMANCE CONSIDERATIONS

When optimizing performance, the goal is to optimize the use of these three critical computer resources: disk I/O, memory and CPU time. For SYBASE, since it has a client/server architecture, there is a fourth resource to consider, namely, network I/O. Reduction in the use of these resources results, in general, in a reduction in elapsed time and an increase in system response time. Another resource that is also important is disk space, although based on SAS user surveys, this consideration is usually less important than...
the first three mentioned. During the discussion of performance issues these topics will be examined:

- subsetting restrictions
- sorting
- use of indexes
- extraction of DBMS data into SAS datasets
- engine locking
- transaction processing
- sequential reading of data
- joining

Use Subsetting Restrictions

Reducing I/O is the most important way to improve the performance of the database engines. By limiting the number of rows and columns that are retrieved from the database, the number of disk I/Os is reduced, fewer internal buffers are allocated so less memory is used, and less code is executed fewer times so CPU cycles are saved. There are six ways to restrict the number of rows that are read by the database engine:

- Enter selection criteria in the view descriptor by using the SUBSET window or the SUBSET statement in the ACCESS procedure. Please note that when doing this, the DBMS column names must be used and the syntax of the clause is not verified until the view is actually read and used by the engine. The WHERE, GROUP BY and HAVING clauses can be used to subset data. An example of a selection clause is

```
WHERE CUSTID > 200
```

- Make use of the SAS WHERE statement when using SAS procedures. In virtually every case, a SAS WHERE statement is translated into a corresponding DBMS WHERE clause and appended to the SELECT statement by the database engine. This is an example of SAS code using a WHERE statement.

```
PROC PRINT DATA=VIEWLIB.USACUST;
WHERE CUSTID > 200;
```

- Use the SAS WHERE data set option after specifying the view descriptor. Data set options can be used after any specification of a SAS data set name. For example,

```
PROC PRINT DATA=VIEWLIB.USACUST (WHERE=(CUSTID>200));
```

- Use the WHERE command in the FSEDIT, FSROWSE, FSVIEW and FSLetter procedures. This command can be used on the command line of the FSEDIT procedure, for example, type WHERE CUSTID > 200 at the command line prompt and only the rows fulfilling this subset restriction are read and displayed by FSEDIT procedure.

- Use the WHERE clause available in the SQL procedure's statements. Again, in virtually every case, an SQL WHERE statement is translated into a corresponding DBMS WHERE clause and appended to the SELECT statement by the database engine. An example of this is

```
PROC SQL; SELECT * FROM VIEWLIB.USACUST
WHERE CUSTID > 200
```

- The OBS=N data set option can be used to limit the number of rows. When this option is used, only the first N rows of the DBMS data set that satisfy the view requirements are displayed. This option is especially valuable when debugging SAS applications.

In addition to reducing the number of rows read from the database, the second way to reduce I/O is to restrict the number of columns read from the database table or view. This is done by following these guidelines:

- Create the view descriptor by selecting only those columns which are required. Any columns that are selected will be read from the DBMS every time the view descriptor is used by the INGRES engine. For the SYBASE engine, every column selected is read except when the view descriptor is used in the following two ways.

For SYBASE, the following additional methods can be used to avoid the expense of reading extra data.

- Use the DROP= and KEEP= data set options when using a view descriptor. With the SYBASE engine, only those columns required to fulfill the DROP= or KEEP= data set options and satisfy any other references to variables in the SAS application code will be read from the DBMS.

- When using the SQL procedure, reference only the SAS variables that you need. For example, instead of using the "" in a SELECT statement, use the actual variable names that you need, if they are fewer than the number of variables selected in the view descriptor. Extra I/O will be avoided because only the columns needed will be extracted from the DBMS.

Scrublitize the Use of Sorting

There are four ways to access data via the engine so that it is retrieved in sorted order. First, use the SORT procedure with the OUT= option to read the data using the view descriptor and create a sorted SAS data set. This is an efficient method to use if the sorted data will be reused several times in the same SAS job. Second, you can store an ORDER BY clause in the selection criteria in the view descriptor; this will ensure that an ORDER BY clause is used in the SQL statement that is passed to the database. A third method is to execute an SQL procedure statement using the ORDER BY clause. Finally, the SAS BY statement will ensure that the data is retrieved in sorted order, since this also causes an ORDER BY clause to be appended to the SQL statement passed to the database. Sorting data is always an added performance cost, even if the most efficient method is used, so avoid unnecessary sorting by scrutinizing ORDER BY and BY processing of your data.

At this point, an example using the SYBASE engine will help illustrate some of the techniques described above. Suppose there are 100 columns and 1000 rows in the table USACUST and the view descriptor VIEWLIB.USACUST is a view of the table USACUST. Only five of the columns were selected when the descriptor was created, plus the clause WHERE CUSTID < 300 was typed in the SUBSET window when the view descriptor was created. If this PRINT procedure statement is executed:

```
PROC PRINT DATA=VIEWLIB.USACUST;
RUN;
```

then 299 rows are selected when this SQL statement is generated by the SYBASE engine:

```
SELECT CUSTID, STATE, CUSTNAME,
ZIPCODE, ADDR FROM USACUST
WHERE (CUSTID < 300)
```

If this PRINT procedure statement were executed:
then the following SELECT statement is generated by the SYBASE engine when it reads the data requested by the PRINT procedure and only 99 rows are selected:

```sql
SELECT CUSTID, STATE, ZIPCODE FROM [SACUST]
WHERE STATE='LA',
KEEP=CUSTID STATE ZIPCODE;
```

So even though five columns were selected when the view descriptor was created, only the three columns in the KEEP= option are read when the descriptor is used. The WHERE= dataset option and the WHERE statement are appended to the SQL WHERE clause and the SAS BY statement was turned into an SQL ORDER BY clause.

**Careful Use of Indexes**

Continuing an examination of performance issues, next look at indexing. Of course, indexing is an essential tool when accessing data from a relational database; the database engines do not tamper with any indexes that have been created on a table or view, nor do they create an index when one would be helpful, and will, therefore, follow the default DBMS action when using any index in selection criteria.

**Extraction of DBMS Data into SAS Datasets**

If you are going to use the same DBMS data in many procedures in the same SAS session, it is advantageous to extract the data into a SAS data set. SAS data sets are organized in a manner that provides optimum access for SAS procedures and the DATA step, so procedures using SAS data set input will frequently take less CPU and I/O time than procedures that use view descriptors to access DBMS information. Also, if the database is being shared by other users on an online system, your program might have a negative impact on the DBMS performance and cause the other users to experience slow response time. A third reason for extracting data is security. If you decide to extract DBMS data into a data set, note that indexing in SAS data sets is available in Release 6.06.

**Effective Use of Engine Locking**

To ensure data integrity, every DBMS system implements locking mechanisms. The database engines honor these locking mechanisms and use them to control contention by multiple users or processes for the same piece of data. To use the database I/O engines effectively as single-user and multi-user data acquisition tools, read and update must be considered. First, we will examine read-only access and then, secondly, locking for update.

Locking that occurs when a view descriptor is used for read access is handled similarly by both the INGRES and SYBASE interface engines. For SAS procedures that require access that is read only, for example, use of the procedures FSBROWSE and PRINT, the data is locked from update by another user only for as long as it takes to read the data pages. After the data is in memory buffers, other users or batch jobs that update the tables will not be locked out. The only drawback to this is that the data that a SAS user is browsing may not be completely up to date, since the data in the original table could be changed while the SAS user is browsing.

For update access, the example we can look at is the use of the FSEDIT procedure. For the FSEDIT procedure, the engine must read a row of data from the DBMS, allow the user to interactively examine the current data and then possibly update it. Two SQL statements must be executed by the interfaces to implement the functionality needed. First, the data must be read via a SELECT statement and then, secondly, the data must be updated using an UPDATE statement.

Neither SYBASE nor INGRES allow records to be read and locked using a record id; therefore, update access via row-level locking is not available. So when the FSEDIT procedure is used on a view descriptor, every page of data that is read is locked from use by other users. This ensures that if an update is required by the the PROC FSEDIT user, their update will have priority and the data will not have been changed after the FSEDIT procedure displayed the data to the user and allowed him or her to edit it.

The DBLOAD procedure creates a table and as it loads the rows into the table, a lock is held on the data until a DBMS COMMIT command is issued. The timing of the COMMIT command is controlled by the COMMIT= statement in the DBLOAD procedure.

**Transaction Processing**

At this time, the SAS user cannot control when the database engine issues a COMMIT command on behalf of the SAS process. For the INGRES engine, the default action is to issue a COMMIT command once at the end of any SAS step. For example, one COMMIT command is issued after any SQL procedure statement and at the end of the reading of data requested by the PRINT procedure. The timing of COMMIT commands during a PROC FSEDIT session are controlled by the FSEDIT AUTOSAVE command; this allows the user to control the number of updates that are executed prior to the issuance of a COMMIT command. The SAVE command can also be typed at any time to commit the data.

Since the default action for SYBASE is to have an implicit COMMIT command after the execution of each SQL statement, the database engine will never issue a commit. Instead several statements will be grouped into one transaction by the engine, if this is necessary.

**Database Engines use Sequential Reading**

Even though a relational database engine may retrieve an observation in the FSEDIT procedure after the observation number is requested on the command line by appearing to execute random access, in actuality the data is always read sequentially. Therefore, it is particularly important for the user of an engine to keep in mind that whenever observation 1000 is requested that observations 1-999 must be read first. The BACKWARDS key in PROC FSEDIT can be used with our engines, but, again, since there is no buffering of data, all observations are reread to find the observation requested via the BACKWARDS key.

**Use Joining Carefully**

One of the powerful uses of relational technology is the ability to join data from multiple data sources. In consideration of performance, consider these two ideas. First, let the DBMS do the work of joining rather than expecting SAS software to do this in the SQL procedures. Doing this allows the DBMS query optimizer to reduce the amount of I/O. If the SQL procedure is used to read more than one view descriptor and perform the join, then all the data from each view must first be read before the join is executed by the SAS System. Second, avoid joining data where a Cartesian product is the result; by using selection criteria with an equality condition to produce the join.

**Additional Tips for Effective Use of SAS/ACCESS Interfaces**

Before concluding this discussion of performance issues, there are a few more tips to remember when using the SAS/ACCESS interfaces:

- Consider dropping and recreating indexes when doing large inserts or updates on your DBMS data. The maintenance
involved in updating the index whenever many rows are inserted or updated may be prohibitive.

- Use caution when changing a DBMS table after descriptors have been created. Any changes to the DBMS table will not be reflected in the existing descriptors and may cause them to be invalid and to convert data incorrectly.

- If the line mode interface to the ACCESS procedure is available, create descriptors using the line mode interface to the ACCESS procedure so that they can be re-created or modified more easily. Writing SAS programs to maintain your descriptors can simplify any re-creation that may be needed.

- Use the ACCESS window to have simultaneous multiple connections to a database. This is helpful if you want to use the FSBROWSE procedure to browse a view while executing the FSEDIT procedure on another view at the same time.

- Remember the cost of data conversion when assigning SAS informat in descriptors. The SAS System processes only two data types: character string and floating point number. Storing your DBMS data in this format and making descriptors to match this data will avoid the cost of data conversion.

CONCLUSION

I hope that the SAS/ACCESS interface architecture, DBMS run-time interface details and performance design strategies described in this paper help you, the user of the SAS/ACCESS interfaces to INGRES or SYBASE, write efficient SAS applications.

REFERENCES


ACKNOWLEDGEMENTS

I would like to thank Christy Kooyman, Howard Plemmons, Emily Wallace and Belinda Weinbrecht for reviewing this paper and offering helpful comments.

SAS and SAS/ACCESS are registered trademarks and SAS/DB2 is a trademark of SAS Institute Inc., Cary, NC, USA.

DB2, SQL and SQL/DS are trademarks of International Business Machines Corporation.

INGRES is a registered trademark of Relational Technology, Inc.

SYBASE is a registered trademark and SQL Server, SQL Toolset, Transact-SQL and DB-Library are trademarks of Sybase, Inc.

VMS and Rdb/VMS are trademarks of Digital Equipment Corporation.