

SAS and DB2: Making the Connection

Kathe Richards, Technical Support Analyst
Texas Comptroller of Public Accounts

Synopsis

How do you find your data, pull it out and put it together when you're a SAS programmer and the data has found a home in DB2? What's the difference between an Access Descriptor and a View Descriptor? And what does PROC SQL have to do with it? Here are tales from a SAS programmer who found answers to these and other questions when the data moved and the rest of us had to follow.

A Little History

Back in the early days of computers, data lived with programs. Remember cards? And CARDS;? You took your program and your data, in the form of a stack of punched cards, over to the computation center, handed it to the guy behind the counter and waited -- maybe a day or two -- for your output, hoping that you hadn't made a keying error. As time and technology progressed, it became possible to construct your program using a terminal and a keyboard and store it somewhere inside the computer. You could store your data the same way, so more than one of your programs could read the same data. Eventually, with the introduction of direct access storage devices and indexed files, we could even have two programs read the same data at the same time. In general, though, we tended to build groups of programs, applications, that owned their own data. Programs and groups of programs tended to cluster around the performance of specific functions. There was a library application and a payroll application and maybe an inventory control application or something else, depending upon what looked to somebody like a good candidate for automation.

As computers became more common and their value became more evident, we began thinking that eventually maybe most information about any particular business or enterprise might be in a computer. We also began noticing that there were overlaps between applications. The same conceptual pieces of data could be found in several places being maintained by several programs (maybe on several different machines) but were called different things. Suddenly, it seemed like a good idea to try to consolidate some of this data. Even with consolidated data files, however, most of the logic required to read, write, build, or delete these files continued to reside in application programs.

The responsibility for handling the housekeeping aspects tended to fall to special purpose utility programs that could do backup and recovery, rebuild indexes, manage available space, and provide some cataloging services. But still, what we had in these data stores was just data. If you dumped it and looked at it, chances are pretty good you wouldn't have much idea about what it stood for.

As computers became exceedingly common and the costs of dealing with them became more and more apparent, it became interesting to hunt for ways of managing this asset. It was postulated that if all the data were in one place, managed by a set of utilities that not only did housekeeping but also undertook a good bit of the work involved in application program access logic and provided some kind of dictionary so we would know what's in there, this would save a lot of scratching around and wheel reinvention by application programmers. Thus, Data Base Management Systems (DBMS) was born.

A database management system is a physical and logical model for managing data and resources. It (theoretically, at least) models, documents, and maintains consistency with the business situation upon which the data is based. And it handles the overhead associated with many of the tedious aspects of managing data (space, access, security, integrity.)

They come in various styles: hierarchical, network, relational, hybrid, and object-oriented are the most common. The most popular these days seems to be relational. A Relational Data Base Management System (RDBMS) is made up of databases that are groups of tables. Each table has rows and columns (also sometimes called tuples and attributes) which are likely to remind SAS users a lot of observations and variables. You could, in fact, think of a SAS data file as a table and a SAS library (assuming you constructed it with that in mind) as similar to a database in an RDBMS. A table contains data that is logically related based upon value. The tables can be matched and merged to select specific subsets and combinations much in the way that SAS data files are. Classically, the information in the tables is managed using a specific language of a small but elegant set of operators called Structured Query Language (SQL). DB2 is IBM's implementation of RDBMS.

In theory, this seems like a really good idea, doesn't it? It's the execution that sometimes gets us in trouble.

In our organization (and probably in many others) the first attempts to convert applications to DB2 involved a technique that I've come to recognize as the "VSAM Plop". The first small new application (or small aging application) that came up for attention became the trial balloon. The data that would normally go into a file went into a DB2 table. We cranked up the application and forged ahead. We discovered that good file design for a specific application was not necessarily good database design. We discovered performance problems, access problems, and maintenance problems. And we did not seem to be discovering the much publicized advantages.

It became apparent that there are some tricks to using this new technology effectively. Among the first things that became evident is that putting the data into tables related by value takes a lot of analysis of the data and how it is used. To keep from having to rebuild massive amounts of application code and restructure databases to respond to each problem that was uncovered, we began to think in terms of a more global solution. We evolved towards the notion that the data is related to the processes that create, read, update and delete them. And the processes are related to the business that we do.

To retrieve some of the advantage of having a general and universal set of data about an enterprise, consistently managed by a single function (the DBMS), somebody has to do a lot of groundwork. This includes things like identifying the basic functions of the business, the processes that support the functions, and the data entities that are involved in the processes. This turns out to be a non-trivial effort. A priesthood of specialists has evolved that will help you do this. If you do not do it carefully and completely, you risk consequences ranging from expensive and time-consuming redesign as the business evolves to not being able to find out where the data is and what it means.

After the business has been modeled, the data identified, and the relationships between the data entities established, we theoretically have the logical framework for building the database. The logical framework is then analyzed by another priesthood of specialists who will decide upon the best physical implementation for the data. Basing the decisions primarily upon the information from the business model regarding what processes use the data in various ways, they apply rules (and sometimes intuition) to turn the entities into tables in databases.

This whole effort, if done correctly and completely, should provide us with a detailed picture of our business, including the information we manage with it's definitions and interrelationships, and some well-structured databases and application programs that are easy to maintain and modify. The promised land will have arrived.

While all this was (and is) going on in our environment, life among the SAS programmers has begun to change. In our organization, SAS programmers serve a very wide variety of purposes. Some do heavy duty statistical analysis. Others do general reporting, often described as "MIS" or "DSS" reporting. Some of this turns out to be programming that would have been (and sometimes was) done in conjunction with some of our production application systems. Some do a lot of file management. Many of our systems programmers use SAS as a kind of general utility. Many divisions have SAS programmers to supply them with services they find they cannot get through the usual channels from the official programming staff.

ACCESS

As data began to be migrated to DB2, the SAS programmers needed to get at that data. SAS provided facilities for doing this. The primary facility, introduced with release 6.06,

is the SAS/ACCESS Interface to DB2. It allows the user to build access descriptors and view descriptors to DB2 tables. The access descriptor builds and saves the logic needed to make contact with a specific DB2 table. The view descriptor describes the selection of data columns and rows the user will get when the view descriptor is executed. Once the access descriptor and associated view descriptor is built, it is referred to in SAS code exactly as if it were a SAS data file. The "Friendly Manual" you will need to describe this facility in detail is "SAS/ACCESS Interface to DB2".

In general, this facility works pretty well, especially as long as you're working with only one table. There are a few minor irritations about it, though. One is the SAS limitation of 8 bytes for variable names. DB2 variable names are much longer and are typically at least somewhat descriptive of the data. In SAS/ACCESS, you get the "real" data names as labels, but you still have to supply SAS variable names or let SAS supply default variable names. If you let SAS supply the variable names by specifying "yes" in the "Assign Variables" field when you are building the access descriptor, you can type over the names that SAS assigns. However, you cannot change the variable names of the view descriptors that you build based upon that access descriptor. We find this to be very inconvenient and therefore usually specify our own names.

Also, if you need to subset the table, you will have to supply the correct WHERE clause, and it must be in DB2, not SAS, syntax. And finally, because DB2 has a specific set of data types doesn't coincide with SAS formats, we needed to be aware of what default formats SAS was assigning. We found that date comparisons for subsetting WHERE clauses in view descriptors required some research in the DB2 manuals. Although your dates will print in MMDDYY8. format by default, you have to use MM/DD/CCYY in your subset WHERE clause.

Building the access and view descriptors is pretty easy. Even though we have a lot of batch users, the Display Manager facility itself didn't present a huge challenge. And with 6.07 we could build view descriptors in batch as WORK. files. Many users preferred to do this and avoid managing libraries of view descriptors.

PROC SQL

When PROC SQL was introduced, some folks thought, naturally enough since SQL is the "official" language with which one communicates to DB2, that this is the facility we needed to read DB2 in SAS. Well, not quite. What PROC SQL really does is allow you to manipulate SAS data files using SQL syntax. This is a very powerful thing to be able to do, especially if you already know how SQL works. (The "Friendly Manual" for PROC SQL is "SAS Guide to the SQL Procedure".)

You can use PROC SQL to communicate with DB2 though that's not really what it was designed for. There are two ways of doing it and it is important to understand what is happening "under the covers" before you choose which way to use.

Since SAS/ACCESS view descriptors can be used just like SAS data files, when you code your PROC SQL statement, the table reference can be a view descriptor name:

```
SELECT * FROM DB2TBL.VIEW1;
```

If DB2TBL.VIEW1 is a view descriptor, this will bring you the contents of the table upon which it is based.

However,

```
SELECT * FROM DB2TBL.VIEW1 A, DB2TBL.VIEW2 B
WHERE A.NAME = B.NAME;
```

will pass all the contents from the table upon which VIEW1 is based and all the contents from the table upon which VIEW2 is based from DB2 to SAS, whereupon SAS will do the match to perform the join. If you have big tables, you can see the problem here.

There is a way to get around this problem. With SQL pass through, you can code the SQL that DB2 is going to understand and imbed it in an SQL statement that SAS is going to understand. The syntax looks like this:

```
PROC SQL;
CONNECT TO DB2 (SSID=DSNQ);
CREATE TABLE WORK.MYTABLE AS
SELECT * FROM CONNECTION TO DB2
(SELECT * FROM DB2_TABLE1 A, DB2_TABLE2 B
WHERE A.NAME = B.NAME);
DISCONNECT FROM DB2;
```

Not intuitively obvious, but it will join the two tables within DB2 and pass the result back to SAS. The syntax of the code within the parentheses must be DB2 syntax. The rest of it is SAS syntax. The reference for this is Technical Report P-221.

Infrastructure

Having the tools was a start. The next challenge was finding the data that had migrated to DB2. We got some help here from the Database Administrators. They volunteered to supply an "official" SAS library of access and view descriptors for all the production DB2 databases and tables. They also supplied a database with a table containing a cross reference between all the tables and the SAS access/view descriptor names and a table containing a cross reference between all the columns in any table and the SAS variable names. The table/descriptor cross reference is invaluable.

The "official" SAS library of descriptors was originally produced because of concerns about security. It has become an enormous convenience for the SAS community and a

headache for the DBAs. As development in DB2 expanded in the agency, several additional DB2 environments have been established to service the needs of development, maintenance, and acceptance testing as well as the production work. The DBAs now attempt to keep libraries available for all of these environments. This process is now somewhat automated.

The security concern was addressed through establishment of a Query DB2 environment, which contains aliases for all production tables that are not maintained within the Query environment. Users can read production tables through the Query environment through a connection established using the Distributed Data Facility (DDF). The DDF was designed to support environments where data resides on different machines, but it has performed adequately as we are using it. We have had a few problems associated with SAS (and QMF as well) jobs that were terminated while the query was being executed in the production environment. The DDF apparently is unaware of the cancellation and the job becomes "hung up" until someone manually issues a command to cancel the DB2 thread.

Gotcha's

With the infrastructure in place, the ACCESS facility from SAS got us up and running. Now we began to see where the real problems were going to occur.

Documentation

The piece of the infrastructure that we did not have was adequate documentation. In most cases, we had made an effort at modeling the data. In some cases, we had limited data dictionary information. We had entity relationship diagrams (ERDs) for the databases that had been built using a CASE tool. The ERD will show the tables in a database, indicate the pieces of data that establish the relationships with other data in other tables, and name the tables and column names. But access to the tools containing the ERD and the data dictionary information is quite limited. A programmer outside the technical support group and the development group that wrote the application would have a hard time getting a copy of the current ERD, simply because they don't have access to the tool.

Even after tracking down a current ERD and whatever data dictionary exists for the function under investigation the programmer may need to spend some time with someone who understands the structure of the application itself to establish to his satisfaction that the data he's looking at is really the data he wants. Some programmers who are sufficiently familiar with the business function of the application are able to figure out what they need by browsing the tables and running some exploratory jobs. In general, though, the new structure of the data, with its many tables and relationships where there was once just one file, has presented an unexpected learning curve, aggravated by the missing infrastructure pieces.

Performance

We are finding that performance is often a very big problem when using SAS with DB2. There are several reasons:

- o There is a lot of overhead associated with an RDBMS. After all, it is managing your data from a global perspective.
- o The data in relational databases is not in any particular order. DB2 overcomes this problem by allowing for indexes to be built on columns that are frequently needed for matching.
- o Often DB2 databases have been designed to optimize processes perceived to be "production" such as transaction processing, at the expense of processes that summarize large amounts of data. This is typically addressed by providing additional tables that support the neglected processes.
- o It takes a pretty good understanding of what is happening and where it is happening to write the SAS code well.

Choosing Strategies

Most SAS programmers have some idea of how to make the code run faster and better. DB2 gives us the opportunity to expand our knowledge in this area. The main consideration here is understanding what part of the work is happening in DB2 and what part is happening in SAS. A bit of documentation that is very worthwhile here is the section in Chapter 3 of Technical Report P-242 entitled "Using SAS/ACCESS Engines Efficiently".

As we might expect, it is especially important when using database data to use as little of it is possible. The more we move stuff around, the more resources it takes. In DB2 this means always specify only the columns you need and the rows you need in the view descriptor you use. When using a view descriptor as an input dataset (in a SET or MERGE statement) use the data set option WHERE= to further limit rows retrieved. Using PROC SQL, don't code SELECT * (and bring back the whole table) when all you need is a few columns. Name the columns.

If you need to match two large tables, try to think of a way to do it within DB2. You could use SQL pass through. You could ask your DBA to build a DB2 view that joins the tables and build your access descriptor and view descriptor on that. You probably do not want to bring all the contents of each table into SAS and make the match within SAS.

The biggest problem that we ran into is the case where we need to match a small number of values in a SAS data file to a DB2 table where the variable is indexed in the DB2 table. There are lots of possible ways to do this, none of which are good under all circumstances:

- o Code it as a straight SAS MERGE. What will happen here is, the entire table will be passed to SAS and sorted. Then the MERGE will take place in SAS. Takes lots of time and space.
- o Code it so that the variable is passed to DB2 as a value. Instead of coding WHERE DB2COL = SASVAR, code WHERE DB2COL = 'Anderson'. If there are more than just a few values, though, this could get very tedious. You could write a macro to do it. (There are several SUGI papers on how to do this - look for Loren and Dickson's paper in SUGI 19, and an example comes with the SAS Sample Library - look for SQLJMAC.) You could write the code to a %INCLUDE file and execute that.
- o If you can get authorization to do it, build a little table of key values within DB2 so you can perform the match within DB2.
- o You can use the MODIFY statement to retrieve the record you need but not actually UPDATE the record. You will have to have update authority on the table to be able to do this.

Debugging

- o We found we could use the EXPLAIN facility in DB2 to figure out how DB2 was going to perform an operation. It's reachable through PROC DBLOAD:

```
PROC DBLOAD DBMS=DB2; SSID=DSNT;  
SQL  
EXPLAIN PLAN FOR SELECT T.ID, T.NAME  
FROM TPINFO.MASTER T, RFDCLM.REFUND R  
WHERE T.ID = R.ID;
```

The results will be written to your PLAN_TABLE.

- o Use OPTIONS \$DB2DEBUG;. This SAS option will print the SQL call generated by your code on your SAS log.
- o Another resource available to help figure out what is going on is DB2 itself. DB2 maintains a database of catalog tables that contain the information DB2 uses to run itself. You can use these tables to get information about other databases in DB2. The database name is SYSIBM and some of the handiest tables are:

SYSINDEXES - One row for every index, including indexes on catalog tables.

SYSKEYS - One row for each column of an index key.

SYSVIEW - Contains the code that generated any particular view.

The information itself is sometimes a little cryptic. The IBM Database2 "Reference Summary" gives some help and the "SQL Reference" is more detailed.

And Some General Recommendations - -

If you're in our position, a SAS programmer whose shop is going DB2, get to know whoever it is that keeps whatever documentation is available. If there is a big new development project under way, keep an eye on it. Just like in the old days, once the new technology is settled in and we understand the new environment, it will probably stay that way for awhile.

Also get to know the DBAs. They manage the database, dispense security authorization, can often make custom DB2 views for you, have all the DB2 manuals at their disposal, and sometimes have great charts and pictures that show the relationships between the system tables.

And look for ways to build some environmental supports and tools to share. This will make everybody's life easier.