A Hierarchical Database Engine for the Version 6 SAS® System
Barbara A. Barrett, SAS Institute Inc., Austin, TX

ABSTRACT
The first Version 6 SAS engine for hierarchical databases was built for SYSTEM 2000® Data Management Software. SYSTEM 2000 software is a hierarchical data management system for mainframe computer systems running under IBM® MVS and CMS host systems. A hierarchical database represents one-to-many relationships among data records at different levels. This paper describes how to use Version 6 of the SAS System to define, load, query, and update a SYSTEM 2000 database.

INTRODUCTION
An engine is a component of the Version 6 SAS System that reads from or writes to a file. Each engine allows the SAS System to access files with a particular format. An interface view engine reads and writes to files that have not been formatted by the base SAS engine. The SAS/ACCESS® interface to SYSTEM 2000 software includes an interface view engine for reading and writing SYSTEM 2000 databases. It also includes the ACCESS procedure for defining a new type of SAS file called a descriptor file, the DBLOAD procedure for creating and loading databases, and the QUEST procedure for using the SYSTEM 2000 languages directly.

SYSTEM 2000 DATABASES
A SYSTEM 2000 database is hierarchical in that you store and access data according to defined relationships between groups of related data. In addition, SYSTEM 2000 databases include indexes, qualification and sorting capabilities, passwords for securing access, both Multi-User® and single-user execution environments, transaction journaling, and rollback recovery.

Advantages of Hierarchical Databases
The main reason to choose a hierarchy to represent information is to gain a performance benefit. The type of application that benefits from using a hierarchical database is one that contains several types of records, each related to the other in some predictable way. As separate files, the different record types contain duplicate information, and are often quite large. But when you put these files in a hierarchical database, you can reduce both the amount of storage required to hold the records and the processing time required to access them. All data management software systems that handle hierarchies are alike in that the software stores information about the data relationships in order to provide the performance advantage. Beyond that, different packages offer different benefits. For example, SYSTEM 2000 software provides item-level security, shared update access, and more.

Some data collections are intrinsically hierarchical. Departments have employees, experiments have results, accounts have transactions, and so on. These are all examples of one-to-many relationships occurring naturally in the data. While all these collections can be stored in multiple relational tables, a hierarchical database can be a very convenient alternative because you do not have to request a value-based join every time you need to hook tables together. The hierarchical database does it automatically.

Using a Hierarchy with the SAS System
The challenge for a view interface engine for a hierarchical database is that the SAS procedures expect to process tabular data. A hierarchy contains not one type of record, but many, and the records repeat in groups that vary in dimension. Furthermore, while each type of record is always related to at least one other type, not every type of record is related to every other type. A number of issues arise. What is an observation? What side effects does updating a single record cause? What does it mean to delete an observation? How do you qualify groups of records? What happens if portions of the relationships are missing? The SAS/ACCESS interface to SYSTEM 2000 software answers these questions in ways that surface the power of the database to the SAS System in a simple and straightforward manner. Consider the following example.

A QUALITY CONTROL APPLICATION
This example is a quality control application. A manufacturing company assembles small power tools. Technicians in the test area monitor a number of variables, as well as track defects. Engineers use the SAS/QC® product to analyze whether or not the process is in control. During each shift, data are collected in SAS data files on microcomputers in the test area. Of course, much of the data are redundant, such as the data that identify the process or product. Once an hour, the collected data are moved to a SYSTEM 2000 hierarchical database on a mainframe.

A quality control analysis looks at either attributes or variables. Attributes are characteristics that either exist or do not, such as whether a belt sander has a good plug or a bad plug. Variables are characteristics that can be measured, such as the current drawn by an electric drill. This application looks at both. Figure 1 shows an outline of the database.

<table>
<thead>
<tr>
<th>Product type</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>Serial number</td>
<td>Hour measured</td>
</tr>
<tr>
<td>Lot number</td>
<td>Measurement type</td>
</tr>
<tr>
<td>Defect</td>
<td>Measurement value</td>
</tr>
</tbody>
</table>

Figure 1  Outline for Quality Control Database

Defining the Hierarchy
The outline of the database reflects the hierarchical structure. Product type and status are at the top of the hierarchy, at level zero. Each product type, like an electric screwdriver, has many serial numbers that are tested, so serial number is at level one. Each serial number may have several defects, so defect data is at level two. The combination of information from level zero to level two is a path in the database. Each product type is measured many times, so measurement data is at level one. Product type plus measurement data is another path in the database, separate from the defect path.
Creating the Database

There are two ways to initially set up the SYSTEM 2000 database with the structure shown in Figure 1. The two methods are the DBLOAD procedure and the QUEST procedure. Either procedure can be run in batch or display manager mode.

With the DBLOAD procedure, you start with a SAS data file, for example, SASDATA.DEFECT, which contains one hour's worth of defect information. The new database is patterned after the input file. You can define a single-path, multilevel database. The DBLOAD procedure loads the input data immediately after creating the database, unless you choose otherwise. Submit the following statements in batch mode to define a three-level database. Other statements are available to rename variables, drop variables, define indexes, and more.

```
      proc dbload dbms=s2k data=sasdata.defect;
         dbn=quality;
         level=serial>lot>defect;
         load;
         run;
```

In display manager mode, the DBLOAD procedure displays two windows. In the first, you identify the database and provide a password. In the second, you indicate those variables on SASDATA.DEFECT that you want included in the database. You also indicate the structure of the database by typing in the hierarchical level of each field. You can define indexes now, or later with the QUEST procedure.

Screen 1 shows the SYSTEM 2000 Load Display Window.

```
SCREEN 1 SYSTEM 2000 Load Display Window

           COMMAND >>
               SYSTEM 2000 Load Display Window

Database: QUALITY

Proc:          Component Name | Index SAS Name | Format
------:          --------------- | -------------- | ------
                    ITEM          | PRODUCT        | $9.
                    ITEM          | STATUS         | $1.
                    ITEM          | SERIAL         | SERIAL RTP12.
                    ITEM          | LOT            | LOT RTP12.
                    ITEM          | DEFECT         | DEFECT $20.

```

Creating Descriptor Files

An access descriptor file is a special type of SAS file created by the ACCESS procedure. The access descriptor is a list of all the items in the database, along with information to access the database. Because SYSTEM 2000 software supports password security, you can also set up different access descriptors for each password, where each password limits the set of items in the descriptor.

A view descriptor is another special type of file created by the ACCESS procedure. View descriptors are the means by which the interface view engine reads and writes to a database. You create a view descriptor using the access descriptor as a blueprint. A view descriptor contains a list of database items, as well as criteria for subsetting database records by value. The person who uses the Quality Control database does not need to know anything about how the database is organized. He just needs to know the name of a view descriptor. You choose whether you require the person to enter a database password.

The DBLOAD procedure creates a default access descriptor when you create a new database. However, you will probably find reasons to create your own, such as to limit access to fields. To create an access descriptor, use the ACCESS procedure.

```
proc access accdesc=s2kdata.quality function=c;
run;
```

If you have more than one SAS ACCESS product, an Engine Selection Window prompts you to select the database management system you want. Put your cursor on SYSTEM 2000 and press ENTER. The Access Descriptor Identification Window, shown in Screen 2, appears next. Fill in the database name, your password, and whether the database is under single-user or Multi-User access, then press ENTER.

Loading Additional Records

You can load records into an existing database either with the DBLOAD procedure or the APPEND procedure. In the following example, collected data are loaded every hour. Both procedures load data through the SYSTEM 2000 view interface engine. All engines operate from special SAS files called view descriptors. A view descriptor contains a list of database items that you want to process with the SAS System. When you define a new database, the DBLOAD procedure automatically creates a view descriptor. Or, you can define your own view descriptor using the ACCESS procedure, described later. The following statements load SASDATA.DEFECT into a database using a view descriptor called S2KDATA.DEFECT. The defect records may be for existing serial numbers or for new ones, or both.

```
      proc dbload dbms=s2k data=sasdata.defect;  <- input
         viewdesc = s2kdata.defect;  <- output
         load;
         run;
```

The QUEST procedure is an alternative way to create a database.

```
proc quest;
   over, demo; new database is quality;
   define;
   1* product (char x(30));
   2* status (non-key char x(5));
   10* products (record);
   10* serial (int 9(5) in 100);
   200* defects (record in 100);
   20* lot (non-key int 9(6) in 300);
   20* defect (char x(20) in 300);
   300* measurements (record in 0);
   300* hour (non-key int 9(6) in 300);
   30* subtype (char x(30) in 300);
   30* awale (non-key dec 9(8).99 in 300);
   map;
```
To create a view descriptor for measurement data, enter

```
proc access accdesc=s2kdata.measure;
run;
```

The View Descriptor Display Window, shown in Screen 4, appears next. It shows all the items in the access descriptor that were not dropped. You can also enter subsetting criteria by entering the command SUBSET. An edit window appears. Enter a WHERE clause, such as

```
where status = production;
```

You can also enter an ordering clause.

**USING YOUR DATABASE WITH THE SAS SYSTEM**

The rest is easy. The DATA step and all the procedures of the SAS System look at your view descriptor as if it were a SAS data file. Not even the SAS procedures know where the data come from. You simply reference a view descriptor with the DATA = option of the SAS procedure. The SAS supervisor and the SYSTEM 2000 interface view engine provide the transparent access. Here is how simple it is to produce a Shewhart chart for our measurement variables.

```
title 'Wobble Measured on Belt Sanders';
proc shewhart data=s2kdata.measure;
   title 'Wobble Measured on Belt Sanders';
run;
```

Output 1 shows the output.

The database can be modified with any of the SAS update procedures, including the APPEND, DBLOAD, FSEDIT, FSVIEW, and...
SQL procedures, and SAS/AF® software. The only SAS programs that cannot update a database are those that create a new output file. The DATA step and the SORT procedure are two examples.

**Delivering Up Observations**

The first job of any interface view engine is to deliver data that look like SAS observations to the procedures. In the case of a hierarchical database, that can be complex. You may choose to define a view descriptor that contains data at several levels in the database. The measurement data portion of the hierarchy lies on a separate path, and is not included. Figure 2 shows a sample data population.

![Diagram of data population](image)

**Figure 2 Sample Population for S2KDATA.DEFECT**

The SYSTEM 2000 engine composes a SAS observation with the four variables using the following logic:

1. Apply any subsetting criteria to provide a list of qualified object records. The object record in this example is the defect record, that is, the record representing the deepest level of detail in the database. The number of object records defines the number of SAS observations.

2. Starting with the first object record in the list, dented case, retrieve the hierarchical ancestors of the object record up through the levels contained in the view descriptor. These are serial number 56399 and electric drill. Present these values as one observation.

3. Move to the next object record, broken switch, and repeat step 2.

Table 1 shows the five observations produced from this process.

<table>
<thead>
<tr>
<th>Obs</th>
<th>Product</th>
<th>Status</th>
<th>Serial</th>
<th>Defect</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>electric drill</td>
<td>prototype</td>
<td>56399</td>
<td>dented case</td>
</tr>
<tr>
<td>2</td>
<td>electric drill</td>
<td>prototype</td>
<td>97123</td>
<td>broken switch</td>
</tr>
<tr>
<td>3</td>
<td>electric drill</td>
<td>prototype</td>
<td>97123</td>
<td>uneven paint</td>
</tr>
<tr>
<td>4</td>
<td>circular saw</td>
<td>production</td>
<td>77521</td>
<td>loose screws</td>
</tr>
<tr>
<td>5</td>
<td>circular saw</td>
<td>production</td>
<td>81146</td>
<td>(missing)</td>
</tr>
</tbody>
</table>

**Table 1 Observations in the View S2KDATA.DEFECT**

**Updating Observations**

Engineers have authority to update the database if a test technician finds a reporting error, or if a process changes status. For example, a tester determines that he entered the wrong serial number for the second electric drill, serial number 97123. It should have been 79123. This value appears in two observations. However, the hierarchical database stores it only once. The engineer wants to update the database using the FSEDIT procedure. Which observation should he change, 2 or 3, or both? The answer is that it does not matter. Since there is only one record in the database for serial number 97123, a change to observation 2 in the FSEDIT procedure will be reflected in observation 3.

**Deleting Observations**

If an update can affect more than one observation, can a delete also do so? The answer is no. Consider Table 2, which shows sample measurement data.

<table>
<thead>
<tr>
<th>Obs</th>
<th>Product</th>
<th>Hour</th>
<th>Mtyp</th>
<th>Mvalue</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>grinder</td>
<td>10</td>
<td>motor torque</td>
<td>7.11</td>
</tr>
<tr>
<td>2</td>
<td>grinder</td>
<td>10</td>
<td>motor torque</td>
<td>7.29</td>
</tr>
<tr>
<td>3</td>
<td>grinder</td>
<td>10</td>
<td>motor torque</td>
<td>7.11</td>
</tr>
<tr>
<td>4</td>
<td>grinder</td>
<td>10</td>
<td>motor torque</td>
<td>7.29</td>
</tr>
<tr>
<td>5</td>
<td>circular saw</td>
<td>11</td>
<td>current drawn</td>
<td>3.39</td>
</tr>
<tr>
<td>6</td>
<td>circular saw</td>
<td>11</td>
<td>current drawn</td>
<td>3.27</td>
</tr>
</tbody>
</table>

An engineer notices a sample size on a SAS/QC chart doubled, and upon investigation discovers that the 10:00 data were accidentally sent to the mainframe twice. Unfortunately, the duplicate data are already in the database. Using the FSVIEW procedure with S2KDATA.MEASURE, the engineer deletes observations 3 and 4. The display immediately shows missing values for all the variables. However, the grinder product type record is still in the database, because other records are anchored to it. If no other records were anchored to grinder, it too would be deleted.

**Inserting New Records**

When you insert a new observation into a hierarchical database, the challenge for the engine is figuring out how many database records should be inserted, and where. For example, a new defect record may represent a new serial number, or it may be another defect on a serial number you are already testing. The Interface view engine determines its actions in several ways. First, it tries the obvious solution. If you are looking at, or positioned to, one observation, does the new observation have any data in common with it? If only the data in the object record have changed, then the insert is unambiguous. A single new object record is inserted, and is anchored to the ancestor that was already there. This action emphasizes the hierarchical tendency to reduce redundancy whenever possible.

But inserts can also be ambiguous. Data above the object record may have changed from the prior observation. The SYSTEM 2000 engine provides an optional facility to resolve ambiguous inserts and still minimize redundant data. It is called a BY key, and you define it in the ACCESS procedure. Screen 5 shows how you designate items in a BY key.

```
ACCESS: Create Descriptor
Command ---
| 1 ACCESS: Create Descriptor | 1 |
| 2 SYSTEM 2000(r) View Descriptor Display Window | 2 |
| Description: | 3 |
| 3 Data set: Library: K2DATA Number: DEFECT Type: VIEW | 4 |
| 4 Output SAS data set: Library: K2DATA Number: View | 5 |
| 5 Database: *DEFECT* Reserved to save: Multi-Value(Tes) The | 6 |
| *DEFECT* ProcSel iSel C| 7 |
| 1 - = 2 C2 FILTERLEN | 8 |
| 1 | 2 C2 FILTERLEN | 9 |
| 1 | 3 | 4 C2 FILTERLEN | 10 |
| 1 | 3 | 5 C2 FILTERLEN | 11 |
| 1 | 3 | 6 C2 FILTERLEN | 12 |
| 1 | 3 | 7 C2 FILTERLEN | 13 |
| 1 | 3 | 8 C2 FILTERLEN | 14 |
| 1 | 3 | 9 C2 FILTERLEN | 15 |
| 1 | 3 | 10 C2 FILTERLEN | 16 |
| 1 | 3 | 11 C2 FILTERLEN | 17 |
| 1 | 3 | 12 C2 FILTERLEN | 18 |
```

**Screen 5 Designating Items in a BY Key**

![Screen 5](image)
A BY key is like a super-match value. It is typically composed of items from each level in the view descriptor, such that the concatenation of the items uniquely identifies a path in the database. When you define a BY key, the SYSTEM 2000 engine goes to the database to search for matches. When successful, one or more new records are attached to the matching ancestors. Again, this action aims for minimum redundancy in the database. Without a BY key, results are less predictable.

Locking the Database
Locking is a way of holding information constant so that it does not change unexpectedly. A SAS procedure can request locks on individual records, on library members, and other types of locks.

The SYSTEM 2000 engine responds to a record lock request by locking all the records that compose one observation. However, the locks do not always succeed. In a hierarchy, records fan out into other records, which fan out into even more records. With Multi-User access, there can be great contention for the records at the top of the fan-out. In order to reduce the contention, the SYSTEM 2000 engine does not require that all the individual record locks succeed for an observation. It does, however, require that a record lock be obtained if an update to the record occurs. If the lock failed when the observation was first retrieved, the lock is retried and the data are compared to what was initially retrieved. The lock must succeed and the data must match before the update occurs. All this happens in a transparent fashion.

For any request stronger than a record lock, for example a library member lock, the SYSTEM 2000 engine puts an exclusive hold on the entire database. An exclusive hold on a database locks out all other users on all paths in the database.

Subsetting Your Database
You can enter a subsetting WHERE clause either in your view descriptor or when you execute a SAS procedure. If you enter the WHERE clause in both places, the two clauses are connected with AND, and processed as one. In the view WHERE clause, you can enter any valid SYSTEM 2000 WHERE clause syntax. In the SAS WHERE clause, you can enter any valid SAS WHERE clause syntax. The syntaxes are similar, but not identical. Each has capabilities the other does not.

The SYSTEM 2000 syntax has the capability to express conditions about the hierarchical structure itself. Although you can only select items from one path, you can qualify on items in any path. For example, qualify those product types that have both a scratched case defect and a case hardness measurement below 25. You cannot express that request with a SAS WHERE clause because it references a variable (MTYPE) that is not known to the SAS procedure.

On the other hand, the SAS syntax includes extensive pattern matching capabilities and arithmetic expressions that are not available in SYSTEM 2000 syntax. For example, qualify those product types where the measurement variable type is 'plating thickness' and the measurement value multiplied by 1.5 is greater than 10.

Fortunately, you can take advantage of the capabilities of both systems. The two WHERE clauses are connected with AND, then the SYSTEM 2000 engine analyzes the entire clause. It resolves minor syntax differences, such as, for range conditions, where SYSTEM 2000 software uses SPANS and the SAS System uses BETWEEN. It sends to SYSTEM 2000 software only those conditions that SYSTEM 2000 software will understand. Any conditions that are sent to SYSTEM 2000 software take full advantage of the indexes in the database. A performance benefit results because fewer records are retrieved from the database into the SAS System. If there are any conditions left, the SYSTEM 2000 engine flags the view to do real-time monitoring of quality data as it is being collected on the microcomputer SAS data files. Engineers can use the same program for analysis on the mainframe SYSTEM 2000 hierarchical database.

Secondly, because SAS data views and SAS data files are alike as far as SAS procedures are concerned, you can develop one program that works on either a view or a file. You can write one SAS program for both technicians and engineers. Technicians can use it to do real-time monitoring of quality data as it is being collected on the microcomputer SAS data files. Engineers can use the same program for analysis on the mainframe SYSTEM 2000 Hierarchical database.

Finally, you can use the DBLOAD procedure to create or load a SYSTEM 2000 database from another database. The input view descriptor can be a SAS file, a view descriptor for another SYSTEM 2000 database, or a view descriptor from another SAS/ACCESS database interface product.

EXTRACTING DATA
The Version 5 SAS interfaces to databases included a SAS procedure to extract data into a SAS file. That capability still exists with Version 6, and can be accomplished in a number of ways. The ACCESS procedure can extract data. A SAS DATA step that reads
a view descriptor and sets it (using a SET statement) to an output SAS data file performs an extract. The OUTPUT command in the FSVIEW procedure accomplishes the same. SQL applications can read views and write files. There is a performance issue to consider. Reusing a view descriptor in consecutive SAS programs implies repetitious qualification, sorting, retrieval, and reformatting of data. If you plan to access the same view descriptor repeatedly in a long-running batch report job, it is usually cheaper to extract the data than to reuse the view descriptor. Of course, once you extract the data, they no longer reflect the actual database or any updates that occur.

CONCLUSION

The introduction of interface engines in Version 6.06 of the SAS System means greater freedom from database details. A hierarchical database, whose structures and hidden pointers are complex, was made to function transparently under the SAS System. The SYSTEM 2000 interface view engine chose specific alternatives for handling observation composition, updates, and data qualification. These choices made it possible to provide additional benefits to SAS applications: rapid access, security, Multi-User access, and automatic error recovery.


IBM is a registered trademark of International Business Machines Corporation. DB2 is a trademark of International Business Machines Corporation.