Using SAS/CONNECT™ and SAS/ACCESS® Software to Access Data In a Distributed Environment

Stephanie K. Creech, SAS Institute Inc., Cary, NC
Jana D. Van Wyk, SAS Institute Inc., Cary, NC

ABSTRACT

SAS/CONNECT™ software offers SAS® users connectivity between numerous operating systems and hardware configurations, and SAS/ACCESS® software allows SAS users access to data in numerous database management systems (DBMS) on a variety of operating systems. When these products are used together, you can have effective remote SAS processing of the data in a database management system. You can also easily transfer the data from the DBMS between the SAS sessions on the local and remote machines.

This paper is intended for users who will benefit from using SAS/CONNECT software and SAS/ACCESS software to develop applications using one or more database management systems in a distributed environment.

INTRODUCTION

The growth of distributed computing environments that use database management systems has made interoperability among data and hardware resources increasingly important. The tools utilized to access and analyze this data across the network are what makes the interoperability possible. Release 6.06 of the SAS System offers two powerful software products, SAS/ACCESS software and SAS/CONNECT software, that together provide an integrated environment for distributed applications. The access to data provided by these software products is twofold: one, the SAS/CONNECT software performs the network operations and, two, the SAS/ACCESS software provides the DBMS communications. By utilizing the remote submission and data transfer features of SAS/CONNECT software, together with the access to DBMSs provided by SAS/ACCESS software, we have a simple, yet flexible, tool to conveniently access databases across a variety of hardware platforms.

The following application will be used to illustrate SAS/CONNECT and SAS/ACCESS software:

A company franchises a chain of bookstores. The district warehouse for the company has a DB2® database management system, running under the MVS operating system, that contains information about the purchasing and inventory needs of the warehouse. Individual bookstores in the franchise maintain an Rdb/VMS™ database management system, under the VMS operating system, to record its inventory and sales data. How can someone making decisions about how much stock to purchase for this warehouse and for each bookstore use their desktop OS/2® workstation to access both of these sources of data and perform analysis and reporting of the data?

The solution is to use SAS/CONNECT software to establish connections from the SAS session on OS/2 to the remote SAS sessions running on MVS and VMS, remotely execute the SAS/ACCESS software to read and update the tables in the databases, and transfer the DBMS data between these systems. Following an introduction to SAS/CONNECT and SAS/ACCESS software, this paper provides a detailed description of this application.

Introduction to SAS/CONNECT Software

SAS/CONNECT software is a cooperative processing product that allows a local SAS session to establish conversations with one or more remote SAS sessions, usually on different machines and operating systems. This gives the user access to data and computing resources on multiple hardware platforms, regardless of their physical location.

The following chart displays the possible connections currently supported by SAS/CONNECT software.

<table>
<thead>
<tr>
<th>Remote Host</th>
<th>AOS/VS, CMS, MVS, PRIMOS*, or VMS*, Release 6.06 and 5.18</th>
<th>UNIX®, Release 6.07</th>
<th>VSE, Release 5.16</th>
</tr>
</thead>
<tbody>
<tr>
<td>OS/2®, Release 6.06</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>UNIX®, Release 6.07</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>PC DOS, Release 6.04</td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>

In addition to connectivity, SAS/CONNECT software offers two other capabilities: remote processing and data transfer. By providing remote submission capabilities, SAS/CONNECT software enables the user to execute SAS programs on the remote system with all output and messages displayed in the local session. SAS/CONNECT software also provides file transfer capabilities. SAS data sets, SAS catalogs and external files can all be transferred across various hardware platforms, giving you the ability to combine data from seemingly incompatible systems. The ability to transfer files and distribute processing to the appropriate host maximizes the utilization of your computing resources.

SAS/CONNECT software includes these SAS procedures and commands:

- The SIGNON command or statement initiates the connection between a local SAS session and a remote SAS session.
- The RSUBMIT command or statement executes the locally entered SAS statements on the remote system, and all generated log and output messages are sent back to the local session for display.
- The UPLOAD procedure copies a file stored on the local system to the remote system.
- The DOWNLOAD procedure copies a file stored on the remote system to the local system.
- The SIGNOFF command or statement terminates the connection between a local SAS session and a remote SAS session.

Introduction to SAS/ACCESS Software

SAS/ACCESS software interfaces between the SAS System and a variety of database management systems. The SAS/ACCESS interfaces currently available allow the SAS System to interface with these database management systems: DB2 and SYSTEM 2000®, under the MVS operating system, Rdb/VMS under the VMS operating system, ORACLE® under the VMS, AOS/VS and PRIMOS® operating systems, SOLIDS® under the CMS operating...
system and INFORMATION under the PRIMOS operating system.

By using the SAS/ACCESS interfaces, a SAS application developer can perform the following tasks:

- Create SAS/ACCESS descriptor files using the ACCESS procedure.
- Read data from the DBMS using the descriptors that were created.
- Create and load a table in the DBMS using the DBLOAD procedure.
- Delete, update or insert data in tables using the DBLOAD procedure's SQL statement, the SQL procedure, SAS/FSP software or the APPEND procedure.
- Create a SAS data set that contains the DBMS data using the ACCESS procedure, the DATA step, the Sal procedure or other SAS statements.

Each SAS/ACCESS interface for the relational databases includes the following three interface tools:

- 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 data 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 DBMS table or a SAS SQL view. (The DBLOAD procedure is not available with all database interfaces.)

The power of the SAS/ACCESS interfaces comes primarily from accessing 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 such things as 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 use a table in a database and storing this information in 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 when data is accessed. View descriptors are created from access descriptors, and each view descriptor can define all or any subset of the data described in the access descriptor.

One important performance issue to consider when creating view descriptors involves deciding how many columns and rows to extract from the database table. Restriction of the number of columns is done by using the SELECT statement to select only the columns that are desired, instead of selecting all of the columns in the table. Restriction of the number of rows is done by specifying selection criteria using the SQL WHERE clause in the SUBSET statement.

The DBLOAD procedure creates and loads a table in an existing database and provides a mechanism to allow the user's ad hoc SQL statements (except a SELECT statement) to be sent to the database. The DBLOAD procedure can load a table using data from a SAS data set, an SQL data view or data described by a view descriptor.

A diagram illustrating the hardware and network configuration used by our application:

This application was designed to accommodate the needs of the purchasers who wish to use their OS/2 workstations to analyze the data residing in the Rdb/VMS and DB2 database management systems so that they can make forecasting decisions, such as how many new books to order for the next month's inventory.

Description of Databases and Tables

Our application uses two databases. The first database is an Rdb/VMS database named SFINVENTORY, which contains inventory and sales data for the bookstore. The second database is a DB2 database, which records the inventory of the district warehouse.

Two tables in the Rdb/VMS database are used in the application. The first is MONTHLYSTKRP, and it holds the monthly report for the amount of books that were sold and that are still on the shelf.
A few of the rows in MONTHLY-STKRPT are:

<table>
<thead>
<tr>
<th>BOOKID</th>
<th>NUMSOLD</th>
<th>NUMSHELF</th>
</tr>
</thead>
<tbody>
<tr>
<td>14569877</td>
<td>253</td>
<td>18</td>
</tr>
<tr>
<td>14898029</td>
<td>27</td>
<td>2</td>
</tr>
<tr>
<td>27654351</td>
<td>52</td>
<td>11</td>
</tr>
<tr>
<td>57673500</td>
<td>1</td>
<td>57</td>
</tr>
</tbody>
</table>

The BOOKID column is a unique key for each book in the store, NUMSOLD contains the number of books sold during the preceding month, and NUMSHELF gives the number of books still on the shelf for this particular book. The second table is named MASTERBOOKS and holds the title, author and so on for each bookid. A few of the rows are:

<table>
<thead>
<tr>
<th>BOOKID</th>
<th>TITLE</th>
<th>AUTHOR</th>
</tr>
</thead>
<tbody>
<tr>
<td>14569877</td>
<td>Sushi, Anyone?</td>
<td>Yokomoto, Akiko</td>
</tr>
<tr>
<td>14898029</td>
<td>Onions, Leeks and Garlic</td>
<td>O'Leary, Michael</td>
</tr>
<tr>
<td>27654351</td>
<td>The Gourmet Microwave</td>
<td>Ringer, Albert</td>
</tr>
<tr>
<td>57673500</td>
<td>Gastronomic Treats</td>
<td>Dull, Anne</td>
</tr>
</tbody>
</table>

Again, BOOKID is a unique key for the table, and the other columns are self-explanatory.

The DB2 database is named CAINVENTORY and is used to manage the inventory of the California district warehouse. There are two tables and one view of interest to our application. The first table is named STKORD and it contains information about the number of books in the warehouse. Three of its rows are:

<table>
<thead>
<tr>
<th>BOOKID</th>
<th>NUMSTK</th>
</tr>
</thead>
<tbody>
<tr>
<td>14569877</td>
<td>534</td>
</tr>
<tr>
<td>14898029</td>
<td>56</td>
</tr>
<tr>
<td>27654351</td>
<td>143</td>
</tr>
</tbody>
</table>

BOOKID is a unique key for each book in the warehouse and NUMSTK is the number of books in stock for each BOOKID. The second table is named MASTERSTOREID; the data in this table describes the books that each store stocks, and four of its rows are:

<table>
<thead>
<tr>
<th>STOREID</th>
<th>BOOKID</th>
</tr>
</thead>
<tbody>
<tr>
<td>1234</td>
<td>14569877</td>
</tr>
<tr>
<td>1234</td>
<td>14898029</td>
</tr>
<tr>
<td>1234</td>
<td>27654351</td>
</tr>
</tbody>
</table>

STOREID is the identification number for a bookstore, and the BOOKID column corresponds to the BOOKID column in the first table. For each store, there is a row in this table for each book that this store sells.

The view named VSTKORD is made by joining the tables STKORD and MASTERSTOREID using equality between the BOOKID columns in each table. This view shows the STOREID, BOOKID and NUMSTK for each STOREID and BOOKID combination that is possible for this warehouse.

<table>
<thead>
<tr>
<th>STOREID</th>
<th>BOOKID</th>
<th>NUMSTK</th>
</tr>
</thead>
<tbody>
<tr>
<td>1234</td>
<td>14569877</td>
<td>534</td>
</tr>
<tr>
<td>1234</td>
<td>14898029</td>
<td>56</td>
</tr>
<tr>
<td>1234</td>
<td>27654351</td>
<td>143</td>
</tr>
</tbody>
</table>

To decide how many books to order for each bookstore, as well as for the warehouse, the purchaser must look at the individual needs for each bookstore in the franchise. The general rule of thumb is to order, for each title in each bookstore, ten percent more books than were sold the previous month. Using the month of August as an example, if 100 books were sold for BOOKID number 12345678, then 110 books should be on the shelf in the bookstore for the month of September. This means that books should be ordered if the number of books on the shelf is less than 110. As each individual bookstore is considered, the warehouse purchasing needs are also calculated.

Data must be acquired from both databases in order to do this analysis. The rows from table MONTHLY-STKRPT in the RdbNMS database should be read if the value of NUMSOLD multiplied by 1.1 is greater than NUMSHELF. The rows from the view VSTKORD are read if they correspond to the bookstore that is currently being analyzed, that is, if the STOREID is equal to 1234, the identification number of the bookstore in San Francisco.

THE STEPS OF THE APPLICATION

The application uses these database tables to illustrate the steps involved in a distributed application using SAS/CONNECT and SAS/ACCESS software to access our data across machine environments. After the SAS session on OS/2 is initialized, the following steps are needed to implement our distributed database application:

- Establish multiple connections between the local OS/2 SAS session and the remote MVS and VMS SAS sessions by executing the SIGNON command for each remote host.
- Create SAS/ACCESS access and view descriptors for each database table or view that will be read.
- Transfer data from the DB2 and RdbNMS databases to the local OS/2 system using the DOWNLOAD procedure.
- Using the data on the local OS/2 system, merge, browse, and derive new data to determine warehouse and bookstore purchasing needs.
- Transfer warehouse and bookstore orders to the remote MVS SAS session using the UPLOAD procedure.
- Add or update the data in the DB2 DBMS on the remote MVS system by using the SQL procedure or the DBLOAD procedure.
- Terminate the connections between the local and remote sessions by using the SIGNOFF command.

In the following sections, each of these steps are described in detail.
Establishing Remote Sessions

First, our application must gain access to the data residing on the remote MVS and VMS systems by utilizing SAS/CONNECT software's ability to establish multiple remote connections. By using the SAS/CONNECT SIGNON statement, the local OS/2 SAS session can establish a connection to each of the remote SAS sessions. The following information is utilized by the SIGNON statement during execution:

- the script file to execute
- the remote session name to initiate
- the communications access method to utilize

The SIGNON statement executes a script file when initiating the connection between the local SAS session and the remote SAS session. A script file is an external file located on the local system that contains a specialized set of SAS statements, called script statements. These script statements provide instructions needed to control the link as well as to start the SAS session in the remote environment. Most SAS/CONNECT software users need to know very little about scripts because they can use the sample scripts provided by SAS Institute; however, the scripts can be modified or rewritten if necessary. If no script file name is specified on the SIGNON statement, the script associated with the default fileref RLIN K will be executed. SAS/CONNECT software also needs to know the name of the remote session to initiate, as well as what communications access method to use. This information can be provided by using the REMOTE= and COMAMID= system options, respectively.

The necessary information is provided, and the connection between the local OS/2 and remote VMS SAS session is established, by executing the following statements on the local system:

```sas
FILENAME RLINK 'CONNECTVMS.SCR'; OPTIONS COMAMID=RASYNC REMOTE=COM1; SIGNON;
```

By defining the default fileref RLINK, the script file VMSSCR is executed by the SIGNON statement. The communications access method and the remote session id are used in establishing the connection to VMS. Because we will be accessing VMS over an asynchronous line attached to the PS/2's standard communications port, COM1, the values of RASYNC and COM1, respectively, are used.

Next, the remote MVS SAS session as well as the connection between the local OS/2 and remote MVS SAS session are established by executing the following statements on the local system:

```sas
FILENAME RLINK 'CONNECTTSO.SCR'; OPTIONS COMAMID=EHLLAPI REMOTE=MVS; SIGNON;
```

Again, the default fileref RLINK is defined so that the script file TSO SCR is executed by the SIGNON statement. The remote session id and the communications access method are specified as they are used in establishing the connection to MVS. The remote session id, MVS, is the short name of the 3270 host session as it was configured with PS/2's Communications Manager. The communications access method used in establishing the connection is EHLLAPI.

Creating the SAS/ACCESS Descriptors

After the connections to both MVS and VMS have been established by SAS/CONNECT software, the next step is to use the SAS/ACCESS software to create descriptors for each table or view in a database from which you wish to read or update data. In Release 6.06, the SAS/ACCESS interfaces allowed creation of descriptors through the full screen Interactive use of the ACCESS procedure only. In Release 6.07 of some SAS/ACCESS interfaces, there will be an additional method of descriptor creation available: the ACCESS procedure will support line mode syntax, which can be used in interactive or batch mode. If the full screen Interactive mode of the ACCESS procedure is used to create descriptors, you should create descriptors by using the SAS system on the VMS or MVS system, that is, without SAS/CONNECT software.

The examples that we describe here illustrate the use of the ACCESS procedure using line mode processing to create the access and view descriptors for all of the DBMS tables that are used in our application. This strategy is used with SAS/CONNECT software to create descriptors at both remote hosts by remotely submitting the ACCESS procedure statements from the local OS/2 workstation.

The following SAS code is sent to the VMS host for execution. When this code executes, the ACCESS procedure creates an access descriptor in the RDBACC library with the name STKAPT, and also creates a view descriptor in the RDBVIEW library with the name VSTKRP T.

```sas
RSUBMIT COM1;
PROC ACCESS DBMS=RDB AD=RDBACC.STKAPT
   FUNCTION=C;
   DATABASE="SFINVENTORY";
   TABLE="MONTHLYSTKRPT";
   SELECT BOOKID NUMSOLD NUMSHELF;
   SUBSET WHERE (NUMSOLD*1.1>NUMSHELF);
   SAVRAS RDBVIEW.VSTKRPT.VIEW;
RUN;
ENDRSUBMIT;
```

These statements are shown bracketed by the SAS/CONNECT statements RSUBMIT and ENDRSUBMIT. When these statements are submitted on the local processor, SAS/CONNECT software performs a remote submission to the SAS session on the remote system. Log and procedure output from SAS programs are displayed by the local session, even though the program executes on the remote system. The optional argument specified on the RSUBMIT statement is the remote session id; this indicates where you wish to execute the SAS statements. COM1 was assigned as the name of the remote VMS session during SIGNON.

The SELECT and SUBSET statements in the code above need further explanation. The SELECT statement is used to choose which columns in the access descriptor to include in the view descriptor. In this example, BOOKID, NUMSOLD and NUMSHELF are selected so that the data corresponding to these columns can be read when the view descriptor is used. The SUBSET statement is used to add selection criteria to the view descriptor. The WHERE clause will be appended to the SQL SELECT statement that is used to extract data from the DBMS.

When analyzing this data, the purchaser wants to look at the information corresponding to the books that are considered low in stock. Books must be ordered if the current number of books on the shelf is less than the number of books sold in the last month plus a ten percent scaling factor. The following WHERE clause satisfies this condition.

```sas
WHERE NUMSOLD * 1.1 > NUMSHELF
```

575
The following SAS code executes on the MVS session and creates an access descriptor in the DB2ACC library with the name STKORD, and also creates a view descriptor in the DB2VIEW library with the name VSTKORD.

```sas
RSUBMIT MVS;
PROC ACCESS DBMS=DB2 AD=DB2ACC.STKORD
FUNCTION=C;
TABLE=BOOKS.STKORD;
SELECT BOOKID NUMSTK STOREID;
SUBSET WHERE STOREID=1234;
SAVEAS DB2VIEW.VSTKORD.VIEW;
RUN;
ENDRSUBMIT;
```

The columns BOOKID, NUMSTK and STOREID are selected for placement in the view, and the WHERE statement that is used causes only the rows corresponding to the store whose STOREID is 1234 to be read by this view descriptor.

Transferring Data from the Remote Systems to the Local System

Once the database view descriptors have been created on the remote systems, they can be used to transfer the database information to our local OS/2 SAS session. The DOWNLOAD procedure copies a view of the database table from the remote host to the local host. By executing the following statements, the RdbVMS STKRPT table information is transferred to the local data set STKORD in the WORK library.

```sas
RSUBMIT COM1;
PROC DOWNLOAD DATA=RDBVIEW.VSTKRPT
OUT=STKORD;
RUN;
ENDRSUBMIT;
```

The power of this technique is that there is no data duplication on the remote VMS host. Since the procedure references a view of the table in the database, there is no intermediate copy of the data needed on the VMS system. The data is read directly from the Rdb/VMS database, transferred over the network and copied into a temporary SAS data set on the local OS/2 session. And, additionally, for efficiency, the view descriptor is created so that only the pertinent data is extracted from the table.

After downloading the data from the Rdb/VMS database, the information from the DB2 database can be transferred. The following use of the DOWNLOAD procedure copies the data referenced by the DB2VIEW.VSTKORD view descriptor to the local system.

```sas
RSUBMIT MVS;
PROC DOWNLOAD DATA=DB2VIEW.VSTKORD
OUT=STKORD;
RUN;
ENDRSUBMIT;
```

DB2VIEW.VSTKORD is a view descriptor for the DB2 view VSTKORD. This is a view of the tables STKORD and MASTERSTOREID, so the data from these tables is transferred to the local data set STKORD in the WORK library.

When the DOWNLOAD procedure is executed on the remote system, the portable engine supervisor reads the view descriptor and loads the appropriate database engine. The engine builds an SQL SELECT statement, passes it to the DBMS system and receives 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.

Local Processing

Now that the information about the inventory of the bookstore and warehouse has been transferred to the local OS/2 system, many different types of SAS processing of this data can be done in the local SAS session, including merging the data, formatting reports and deriving new data.

In order to summarize the current inventory status, the purchaser wants to merge the data from the two data sources so that the data common to each unique BOOKID is in the same observation in a SAS data set. This is accomplished by performing the following join in the SQL procedure.

```sql
PROC SQL;
CREATE TABLE JOININFO AS
SELECT STKORD.BOOKID, NUMSHELF, NUMSTK, NUMSOLD, STOREID
FROM STKORD, STKRPT
WHERE STKORD.BOOKID=STKRPT.BOOKID;
RUN;
```

This code creates a SAS data set named JOININFO, in the WORK library, that contains the SAS variables BOOKID, NUMSHELF, NUMSTK, NUMSOLD and STOREID. Since the join was done by using the equality condition on the variable BOOKID, the information in each observation corresponds to a single BOOKID.

The following illustration shows three of the observations that were created by the join.

<table>
<thead>
<tr>
<th>BOOKID</th>
<th>NUMSHELF</th>
<th>NUMSTK</th>
<th>NUMSOLD</th>
<th>STOREID</th>
</tr>
</thead>
<tbody>
<tr>
<td>14569877</td>
<td>18</td>
<td>534</td>
<td>253</td>
<td>1234</td>
</tr>
<tr>
<td>14898029</td>
<td>2</td>
<td>56</td>
<td>27</td>
<td>1234</td>
</tr>
<tr>
<td>27654351</td>
<td>11</td>
<td>143</td>
<td>52</td>
<td>1234</td>
</tr>
</tbody>
</table>

After the data has been joined, the data set can be printed using the PRINT procedure or the REPORT procedure. Or, if the purchaser wants to look at the data, the FSBROWSE procedure can be used to simply view the contents of the data set JOININFO. The following is an example using the FSBROWSE procedure.

```sas
PROC FSBROWSE DATA=JOININFO; RUN;
```

It is also possible to derive new data by using the existing data in the data set JOININFO. This DATA step

```sas
DATA ORDERS(KEEP=STOREID BOOKID);
STOREORD=ROUND(NUMSOLD*1.1)-NUMSHELF;
IF STOREORD>NUMSTK THEN
WAREORD=STOREORD-NUMSTK;
ELSE
WAREORD=0;
RUN;
```

creates a data set named ORDERS, in the WORK library, that contains the variables STOREID, BOOKID, STOREORD and WAREORD. STOREORD and BOOKID contain the same values as were in the JOININFO data set. STOREORD is the number of books that must be ordered by the bookstore to fulfill its inventory.

576
These are just a few examples of the possible uses of the data that was downloaded from VMS and MVS. Using the power of the SAS System, together with its integrated system of software products, enables you to do much more local processing of this data. In Release 6.06 of the SAS System under OS/2, it is also possible to load the data contained in local SAS data sets into a Database Manager, Lotus® or dBASE® system on OS/2.

Transferring Data from the Local System to the Remote System

After the inventory data from the bookstore and the warehouse has been used to predict the new inventory needs, this newly derived data in the data set ORDERS can be transferred to the remote MVS system. By executing the UPLOAD procedure, the local data set ORDERS is copied from the WORK library on the OS/2 system to the data set WORK.ORDERS on the remote MVS system.

Remote Processing to Update the Database

It is necessary to place the new information in a table named ORDERS in the DB2 database so that this table can be queried by the warehouse staff when they want to generate reports about which books are being ordered. After the data has been uploaded to the remote MVS host, this database table can be updated in the DB2 database using several methods. The following two examples show how to insert data into a table in the database. First, if the table already exists, the SQL procedure’s INSERT statement can be used to insert data into the table and, second, the DBLOAD procedure can be used to create a table and insert data into it.

Assuming that a table named ORDERS already exists in the DB2 database on MVS, the following SQL procedure inserts the SAS data set ORDERS into the table.

\[
\text{PROC SQL;}
\text{INSERT INTO DB2VIEW.ORDERS}
\text{SELECT STOREID, BOOKID, STOREORD, WAREORD}
\text{FROM ORDERS;}
\]

The view descriptor DB2VIEW.ORDERS was created earlier and is now used as the target of the insert operation. The SAS variables STOREID, BOOKID, STOREORD and WAREORD are extracted from the SAS data set ORDERS, and by using the SELECT clause, the data corresponding to these four variables is inserted into the table described by the view descriptor DB2VIEW.ORDERS.

Alternatively, if the table ORDERS did not already exist, we can insert the data by using the DBLOAD procedure. This procedure creates the table and inserts the data. There is no view descriptor required when this method is used. Here is an example of a DBLOAD procedure.

\[
\text{PROC DBLOAD DBMS=DB2 DATA=ORDERS;}
\text{SSID=DB2;}
\text{TASK=BOOKS.MONTHLYORDERS;}
\text{LOAD;}
\text{RUN;}
\]

This example creates a table named MONTHLYORDERS with column names STOREID, BOOKID, STOREORD and WAREORD and inserts the contents of the ORDERS data set into this table. After this data has been inserted into the database, the data can be used to generate purchasing requests for the warehouse or individual stores.

Terminating Remote Sessions

Upon completion, the SIGNOFF command is executed in order to terminate the connection between the local and remote SAS sessions. The following statements terminate the remote SAS VMS session as well as the connection.

\[
\text{FILENAME RLINK 'CONNECT\VMS.SCR';}
\text{SIGNOFF COM1;}
\]

The script file VMS.SCR will be used by the SIGNOFF command when terminating the connection to the VMS system. The SIGNOFF command takes the remote session id as an optional parameter. In this example, the value of COM1 is specified so that the connection between the local OS/2 and the remote VMS session will be terminated.

The following statements terminate the remote MVS SAS session and the connection to MVS.

\[
\text{FILENAME RLINK 'CONNECT\TSO.SCR';}
\text{SIGNOFF MV5;}
\]

The script file TSO.SCR will be used by the SIGNOFF command when terminating the connection to MVS. The remote session id, MVS is specified on the SIGNOFF command so that the connection between the local OS/2 and the remote MVS session will be terminated.

EFFICIENCY CONSIDERATIONS

The application described in this paper is one example of a distributed database environment. When you develop an application for your distributed environment, consider these efficiency considerations.

• Remember the cost of copying data over the network. Carefully evaluate when data transfer is necessary. Be sure you copy only the data that is needed by using selection criteria when creating the SAS/ACCESS view descriptors. See the SUGI 15 paper, "Performance Considerations for the Database Engines" and the SUGI 16 paper, "Efficient Use of the SAS/ACCESS Interfaces to INGRES® and SYBASE®" for more details.

• Remember that descriptors should be created only once: when an application is first designed. Re-create the descriptors only when information in the tables or the selection criteria has changed.

• Always consider the effects of database locking rules on your application. In general, there will be no locks on a database table after the data has been extracted from the database. Therefore, other users of this database could be updating the information in the database while the local processing is taking place.

• Security is also an important factor to consider. View descriptors should be created with this in mind and made available to selected users.

• If an application will be used several times, it may be more efficient to place the code in external files and then use the SAS/CONNECT feature of remote submission to INCLUDE these files. These external files that contain SAS code can be stored on the local system or the remote system and included accordingly.

• To automate an application, the SAS/CONNECT software...
information provided in the RLINK literal definition, the COMANDO- and REMOTE- system options and the SIGNON statement, can be placed in a SAS autoexec file that is automatically executed every time a local SAS session is initialized. It is also possible to put the LIBNAME statements that refer to the libraries containing the SAS/ACCESS access and view descriptors in an autoexec file.

• It is possible to execute the SAS/CONNECT and SAS/ACCESS software in batch mode, so these programs may be run during nightly processing. To verify that the data transfer was successful, use the %SYSRPUT macro statement and the SYSINFO macro variable to check the return codes of the DOWNLOAD and UPLOAD procedures. These tools allow nightly jobs to test whether the remote processing was successfully completed before beginning another step on either the remote host or the local host.

• Hardware access time charges can be reduced by using the SIGNON and SIGNOFF statements efficiently. By terminating a session when access to the remote system is not required, you can reduce your access charges.

• The distributed nature of the application can be transparent to the user of an application using SAS/CONNECT if a SAS/AF® front-end is written as a user interface. The AF menu can be used to describe the contents of each data source without revealing that the data is actually stored on and transferred from another system.

CONCLUSION
This paper has shown you how SAS/CONNECT and SAS/ACCESS software can be used to integrate your computing resources and data across different platforms, as well as virtualize database connectivity. Peer-to-peer and multihost capabilities will enable any operating system environment to function as the local or the remote session of a SAS/CONNECT conversation. For example, a local VMS session could initiate remote sessions on an OS/2 workstation or another VMS system. Some of the access methods that are currently being developed and tested by the staff at SAS Institute Inc. will allow SAS/CONNECT software to become a complete peer-to-peer communication product for all platforms that support asynchronous processing.

Appendix 1 - FUTURE OF SAS/ACCESS AND SAS/CONNECT SOFTWARE
The initial 6.07 release of SAS/CONNECT for the UNIX platforms brings with it SAS/CONNECT software’s first step into peer-to-peer connectivity. Peer-to-peer capabilities will enable any operating system environment to function as both the local and the remote session of a SAS/CONNECT conversation. For example, a local VMS session could initiate remote sessions on an OS/2 workstation or another VMS system. Some of the access methods that are currently being developed and tested by the staff at SAS Institute Inc. will allow SAS/CONNECT software to become a complete peer-to-peer communication product for all platforms that support asynchronous processing.

Another feature currently under development is background execution of a SAS/CONNECT session. Currently, SAS/CONNECT software serially performs local and remote processing which prohibits local processing to take place until any remote execution has completed. This future background processing feature will allow the local SAS session to regain control immediately after a remote submit is issued. This will allow the user to continue to perform local processing while remote processing occurs. For example, while the download of a large data set is performed, you may continue to execute procedures and windows in the local session as well as remote submit to other inactive remote sessions. Background processing will not only give you more control over your processing utilization, but it will also provide a way to decrease the amount of time spent completing certain tasks, as simultaneous remote and local processing will be possible.

With Release 6.06 of the SAS System under OS/2, the SAS/ACCESS interfaces to Database Manager, dBASE and Lotus will be available for the first time. Most SAS/ACCESS interfaces that were available with Release 6.06 will have maintenance releases in Release 6.07 of the SAS System. In addition, the new SAS/ACCESS interfaces that will be available with Release 6.07 of the SAS System are: IMS/VMS, ADABAS® and CA-DATACOM® under the MVS operating system and INGRES under the VMS operating system.

Appendix 2 - HARDWARE AND SOFTWARE DETAILS
Following is detailed information about the hardware, software and network connections used in our application:

Hardware and Connections:
IBM PS/2 Model 80 running OS/2
IBM 3270 Connection Card coax attached to a remote 3174 controller
COM1 asynchronous connection using 2400 baud modem
IBM 39090 Model 600 running MVS/ESA
VAXstation 3100 running VMS
The PS/2 is running IBM’s OS/2 Extended Edition 1.2 with Communications Manager installed. Communications manager has been configured to run 4 DFT host sessions. We use one of the 4 sessions for the MVS access. VMS is accessed asynchronously over the 2400 baud modem.

Software Installed:
Base SAS System on MVS, VMS and OS/2
SAS/CONNECT software on MVS, VMS and OS/2
SAS/ACCESS software on MVS and VMS
Rdb/VMS V3.18-0 on VMS
VAX SQL V3.18-0 on VMS
DB2 version 1.3

REFERENCES
ACKNOWLEDGMENTS

We would like to thank Howard Plemmons for helping us design this application; Jeff Crench, Cheryl Garner, Beth Langston, Howard Plemmons, Emily Wallace and Jack Wallace for reviewing this paper and making helpful suggestions; and Richard Smith for presenting this paper at SUGI 16.

SAS, SAS/ACCESS, SAS/FSP and SAS/AF are registered trademarks and SAS/DB2 and SAS/CONNECT are trademarks of SAS Institute Inc., Cary, NC, USA.

ADABAS is a registered trademark of Software AG.

cBASE is a registered trademark of Ashton-Tate Corporation.

DECnet, VAXstation, VMS and Rdb/VMS are trademarks of Digital Equipment Corporation.

IBM and OS/2 are registered trademarks, and DB2 and SQL/DS are trademarks of International Business Machines Corporation.

INGRES is a registered trademark of Relational Technology, Inc.

Lotus is a registered trademark of Lotus Development Corp.

ORACLE is a registered trademark of ORACLE Corporation.

PRIMOS is a registered trademark of PRIME Computer, Inc.

SYBASE is a registered trademark of SYBASE Inc.

UNIX is a trademark of AT&T.