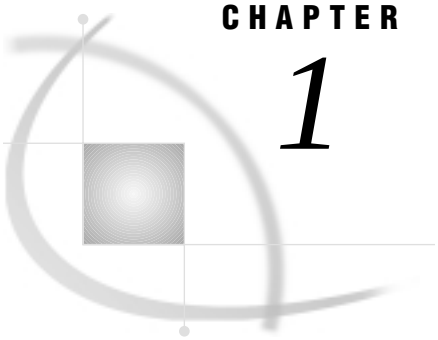


CHAPTER

1

DB2 under OS/390 Chapter, First Edition

Introduction

This chapter introduces SAS System users to Database 2 (DB2), IBM's relational database management system for OS/390. It accompanies and should be used with *SAS/ACCESS Software for Relational Databases: Reference* (order # 57204).*

This chapter describes the SAS/ACCESS LIBNAME and data set options that are specific to DB2. It then focuses on the terms, basic concepts, and options that help you use the SAS/ACCESS interface to DB2. Next, it describes the DB2-specific options and statements that you use in the ACCESS and DBLOAD procedures and in the SQL procedure's CONNECT statement. The statements include those that are necessary to accommodate IBM's addition of DRDA support for DB2. Also documented in this chapter is a Version 5 procedure, DB2UTIL, which enables you to make modifications to a DB2 table using a SAS data set. Finally, there are some tips for improving your performance and some DB2–specific information for the database administrator.
For general information about database management systems, including information for the database administrator about how the SAS/ACCESS interfaces work, refer to Appendix 2, "DBMS Overview and Information for the Database Administrator" .

SAS/ACCESS LIBNAME STATEMENT

This section describes the LIBNAME statement and its options that are specific to DB2. The LIBNAME statement and options that can be used in most databases are fully described in *SAS/ACCESS Software for Relational Databases: Reference*. This section also describes the connection options for DB2 and any DB2–specific LIBNAME options.

SAS/ACCESS LIBNAME Statement: DB2 Specifics

Associates a SAS libref with a DBMS database, schema, server, or group of tables and views.

Valid: Anywhere

Syntax

LIBNAME *libref SAS/ACCESS-engine-name SAS/ACCESS-engine-connection-options* < *SAS/ACCESS-LIBNAME-options*>;

Arguments

libref
is any 8–character SAS name that serves as an alias to associate the SAS System with a database, schema, server, or group of tables and views.

SAS/ACCESS-engine-name
is a SAS/ACCESS engine name for your DBMS, in this case, DB2. SAS/ACCESS engines are implemented differently in different operating environments. The engine name is required.

SAS/ACCESS-engine-connection-options
are options that you specify in order to connect to a particular database; these options are different for each database. If the SAS/ACCESS engine connection options contain characters that are not allowed in SAS names, enclose the values of the options in quotation marks. On some DBMSs, if you specify the appropriate system options or environment variables for your database, you can often omit the SAS/ACCESS engine connection options.

SAS/ACCESS-LIBNAME-options
are options that apply to the objects in a DBMS, such as its tables or indexes. For example, the READ_LOCK_TYPE= option enables you to set a table lock on an operation. Support for many of these options is DBMS-specific.

Some SAS/ACCESS LIBNAME options can also be specified as SAS/ACCESS engine data set options. When you specify an option in the LIBNAME statement, it applies to objects and data that are referenced by the libref. A SAS/ACCESS data set option applies only to the data set on which it is specified. If a like-named option is specified in both the SAS/ACCESS engine LIBNAME statement and after a data set name (which references a DBMS table or view), the SAS System uses the value that is specified after the data set name.

For more information, refer to Chapter 4, "SAS/ACCESS Data Set Options" .

Details The LIBNAME statement associates a libref with a SAS/ACCESS engine in order to access tables or views in a database management system (DBMS). The SAS/ACCESS engine enables you to connect to a particular DBMS and, therefore, to specify a DBMS table or view name in a two-level SAS name.

For example, in MYLIB.EMPLOYEES_Q2, MYLIB is a SAS libref that points to a particular group of DBMS objects, and EMPLOYEES_Q2 is a DBMS table name. When you specify MYLIB.EMPLOYEES_Q2 in a DATA step or procedure, you dynamically access the DBMS table. Beginning in Version 7, the SAS/ACCESS LIBNAME supports reading, updating, creating, and deleting DBMS tables.

To disassociate or clear a libref from a DBMS, use a LIBNAME statement and specify the libref (for example, MYDBLIB) and the CLEAR option as follows:

```
libname mydblib CLEAR;
```

Clearing the libref causes the database engine to disconnect from the database, closing any open plans and releasing any resources that are associated with that connection.

See for more information about options that you can use in the LIBNAME statement.

SAS/ACCESS Engine Connection Options The SAS/ACCESS engine connection options for DB2 are as follows:

SSID= on page 3

SERVER= on page 4

SSID=*DB2–subsystem-id*
specifies the DB2 subsystem ID to connect to at connection time.

SSID= is optional. If you omit it, SAS connects to the DB2 subsystem that was specified in the SAS system option, DB2SSID=. For more information, see “Setting

Your DB2 Subsystem Identifier" on page 14. The DB2-subsystem-id is limited to four characters.

SERVER=*DRDA server*
specifies the DRDA server that you want to connect to. SERVER= enables you to access DRDA resources stored at remote locations. Check with your system administrator for system names. You can only connect to one server per LIBNAME statement.

SERVER= is optional. If you omit it, you access tables from your local DB2 database unless you have specified a value for the LOCATION= option. There is no default value for this option.

Note: Refer to the OS/390 installation instructions for information about configuring SAS to use the SERVER= option. △

DB2 Restriction on Connections The DB2 engine restricts the maximum concurrent open cursors to 32 for a given connection. Note that if you are working with a DB2 table that accesses other tables, you could be opening more cursors than you are aware of.

Beginning in Version 7, there is no limit to the number of connections that you can have to DB2. The DB2 engine uses the Call Attachment Facility (CAF) or the Recoverable Resource Manager Sercvice Attachment Facility (RRSAF) to make an explicit connection to the local DB2 subsystem. The DB2 engine creates one connection to DB2 from the main SAS task. For each subsequent connection to the CAF or RRSAF, the DB2 engine attaches a separate OS/390 subtask.

SAS/ACCESS LIBNAME Options The SAS/ACCESS interface to DB2 supports all of the SAS/ACCESS LIBNAME options listed in Chapter 3, "SAS/ACCESS LIBNAME Statement" , except for DBINDEX=, DBPROMPT=, and DBMAX_TEXT=. In addition to the supported options, the following LIBNAME options are used only in the interface to DB2 or have DB2–specific aspects to them:

This section describes the LIBNAME statement and its options that are specific to DB2. The LIBNAME statement and options that can be used in most databases are fully described in "SAS/ACCESS Software for Relational Databases." This section describes the connection options for DB2 and any DB2-specific LIBNAME options.

AUTHID=*authid*
enables you to qualify your DB2 table names with an authorization ID, user ID, or group ID.

When you specify the AUTHID= option, every table that is referenced by the libref is qualified as *authid.tablename* before any SQL code is passed to DB2. If you do not specify a value for AUTHID=, the table name is not qualified before it is passed to DB2. Once DB2 receives the table name, it automatically qualifies it with your OS/390 user ID. You can override the LIBNAME AUTHID= option by using the AUTHID= data set option.

AUTHID= is limited to 8 characters. This option is not validated until you access a table.

CONNECTION= SHAREDREAD | GLOBALREAD | UNIQUE | SHARED | GLOBAL

enables you to share one connection for reading, updating, and outputting to tables.

In addition to the SHAREDREAD, GLOBALREAD, and UNIQUE values, which are described in , you can also specify SHARED or GLOBAL for CONNECTION= in the SAS/ACCESS interface to DB2. The SHARED value allows you to share one connection for tables referenced by the given libref. The GLOBAL value allows you to share a connection for tables referenced by all librefs for which CONNECTION=GLOBAL is specified.

Use this option with caution. If the connections that are shared are for reading and updating, there is a good possibility that the read cursors will have to be resynchronized if a commit or rollback is performed by the update or output connections. If the cursors are resynchronized, there is no guarantee that the new solution table will match the original solution table that was being read.

You can use CONNECTION=SHARED to eliminate the deadlock that can occur when you create and load a DB2 table from an existing DB2 table that resides in the same database or tablespace. This only happens in certain output processing situations and is the only recommended use for the CONNECTION=SHARED option.

In the following example, DB2DATA.NEW is created in the database TEST. Because the table DB2DATA.OLD exists in the same database, the option CONNECTION=SHARED allows the DB2 engine to share the connection for both reading the old table and for creating and loading the new table.

```
libname db2data db2 connection=shared;
data db2data.new (in = 'database test');
 set db2data.old;
run;
```

In the following example, you have two different librefs that share one connection.

```
libname db2lib db2 connection=global;
libname db2data db2 connection=global;
data db2lib.new(in='database test');
   set db2data.old;
run;
```

If you did not use the CONNECTION= option in the above examples, you would deadlock in DB2 and get the following error:

```
ERROR:  Error attempting to CREATE a DBMS table.
ERROR:  DB2 execute error DSNT408I SQLCODE = --911,
ERROR:  THE CURRENT UNIT OF WORK HAS BEEN ROLLED
        BACK DUE TO DEADLOCK.
```

For a full description of this option, refer to Chapter 3, "SAS/ACCESS LIBNAME Statement" .

IN='*database-name.tablespace-name*'| 'DATABASE *database-name*'

enables you to specify the database and tablespace in which you want to create a new table. The IN= option is relevant only when you are creating a new table. If you omit this option, the DB2 default is to create the table in the default database, implicitly creating a simple tablespace.

'*database.tablespace*' specifies the names of the database and tablespace. Enclose the entire specification in single quotes.

'DATABASE *database-name*' specifies only the database name. Enclose the entire specification in single quotes.

LOCATION=*location*
enables you to further qualify exactly where a table resides.

In the DB2 engine, the location is converted to the first level of a three-level table name: LOCATION.AUTHID.TABLE. The connection to the remote DB2 subsystem is done implicitly by DB2's Distributed Data Facility (DDF) when DB2 receives a three-level table name in an SQL statement.

LOCATION= is optional. If you omit it, SAS accesses the data from the local DB2 database unless you have specified a value for the SERVER= option. This option is not validated until you access a DB2 table. If you specify LOCATION=, you must also specify the AUTHID= option.

PRESERVE_COL_NAMES=YES | NO
preserves spaces, special characters, and case in DB2 column names.

The default value for PRESERVE_COL_NAMES= under DB2 is NO because DB2 converts table and column names to uppercase by default. However, DB2 is a case-sensitive DBMS. To preserve the case of the column names that you send to DB2, use quotation marks around the column names.

PRESERVE_NAMES= is a combination alias for *both* PRESERVE_COL_NAMES= and PRESERVE_TAB_NAMES=. If you set PRESERVE_NAMES=YES, it is equivalent to setting both of these options to YES.

For a full description of PRESERVE_COL_NAMES=, refer to Chapter 3, "SAS/ACCESS LIBNAME Statement" .

PRESERVE_TAB_NAMES=YES | NO
preserves spaces, special characters, and case in DB2 table names.

The default value for PRESERVE_TAB_NAMES= under DB2 is NO because DB2 converts table and column names to uppercase by default. However, DB2 is a case-sensitive DBMS. To preserve the case of the table names that you send to DB2, use quotation marks around the table names.

PRESERVE_NAMES= is a combination alias for *both* PRESERVE_COL_NAMES= and PRESERVE_TAB_NAMES=. If you set PRESERVE_NAMES=YES, it is equivalent to setting both of these options to YES.

For a full description of this option, refer to Chapter 3, "SAS/ACCESS LIBNAME Statement" .

READ_LOCK_TYPE=TABLE
specifies that a SHARED table lock is set on a DB2 table during a READ operation.

READ_LOCK_TYPE= is useful if you want to lock out concurrent changes in order to access an entire table as it is at that particular moment. You can also use it to prevent timeouts from contention with other application processes that are reading the same table.

The default behavior is to allow DB2 to handle the locking process. Consult your DBA to determine what the locking process is for your installation of DB2.

If you set READ_LOCK_TYPE=TABLE, then you must also set the CONNECTION= option to UNIQUE, which means that there is a separate physical connection for each table that is opened in your SAS application. You cannot share a connection when a DB2 table is locked. If you do not set the CONNECTION= option to UNIQUE, the SAS step fails.

If you set READ_LOCK_TYPE=TABLE, you might also want to evaluate whether or not you want to change the SPOOL= option from its default value of YES. If the table is locked and its data cannot be changed during the read transaction, you may not need to create a utility spool file.

Note: Use READ_LOCK_TYPE= with caution because it locks all the tables in a nonsegmented table space, even if they are not the table that is specifically locked. The locks are held until a commit point or until the connection is freed. All

other application processes are locked out of the non-segmented table space for the duration of the lock. △

For a full description of this option, refer to Chapter 3, "SAS/ACCESS LIBNAME Statement" .

SPOOL=YES | NO
specifies whether SAS creates a utility spool file during read operations that require two passes through the table and that are performed with the specified libref.

The default value is SPOOL=YES. For a full description of this option, refer to Chapter 3, "SAS/ACCESS LIBNAME Statement" .

UPDATE_LOCK_TYPE = TABLE
specifies that an exclusive table lock is set on a DB2 table during an UPDATE operation.

If you are updating a large part of a table, you can improve your performance by using UPDATE_LOCK_TYPE= to lock the entire table and prevent other application processes from having concurrent access to it. This is more efficient than locking each page as it is updated and then unlocking it when the changes are committed. You can also use UPDATE_LOCK_TYPE= to prevent timeouts from contention with other application processes that are updating the same table.

The default behavior is to allow DB2 to handle the locking process. Consult your DBA to determine what the locking process is for your installation of DB2.

If you set UPDATE_LOCK_TYPE=TABLE, then you must also set the CONNECTION= option to UNIQUE, which means that there is a separate physical connection for each table that is opened in your SAS application. You cannot share a connection when a DB2 table is locked. If you do not set the CONNECTION= option to UNIQUE, the SAS step fails.

If you set UPDATE_LOCK_TYPE=TABLE, you might also want to evaluate whether or not you want to change the SPOOL= option from its default value of YES. If the table is locked and its data cannot be changed during the update transaction, you might not need to create a utility spool file.

Note: Use UPDATE_LOCK_TYPE= with caution because it locks all the tables in a nonsegmented table space, even if they are not the table that is specifically locked. The locks are held until a commit point or until the connection is freed. All other application processes are locked out of the non-segmented table space for the duration of the lock. △

For a full description of this option, refer to Chapter 3, "SAS/ACCESS LIBNAME Statement" .

Specifying a LIBNAME Statement to Access DB2 Data

In this example, the libref MYLIB uses the DB2 engine to connect to the DB2 database that is specified by the SSID= option with a connection to the remote server, **testserver**. The SAS/ACCESS engine connection options are SSID= and SERVER=.

```
libname mylib db2 ssid=db2
   authid=testuser server=testserver;
proc print data=mylib.staff;
   where state='CA';
run;
```

SAS/ACCESS Data Set Options: DB2 Specifics

This section describes options that can be applied to SAS data sets that access data in DB2 tables and views. In some cases, the option is fully described in Chapter 4, "SAS/ACCESS Data Set Options", except for some DB2-specific detail, such as a default value. In other cases, the entire option is DB2 specific, so it is fully described in this chapter.

When specified in a DATA step or SAS procedure, the following data set options can be used on a SAS data set that accesses data in a DBMS object, such as a table or view. A data set option applies only to the SAS data set, or DBMS object, on which it is specified.

The SAS/ACCESS interface to DB2 supports all of the SAS/ACCESS data set options listed in Chapter 4, "SAS/ACCESS Data Set Options" , except for DBINDEX=, DBPROMPT=, DBMAX_TEXT=, and SASDATEFMT=. In addition to the supported options, the following data set options are used only in the interface to DB2 or have DB2–specific aspects to them:

Note: The DBINDEX= data set option is not supported in the SAS/ACCESS interface to DB2. △

AUTHID=

Enables you to qualify the specified table with an authorization ID, user ID, or group ID.

Default value: None

Syntax

AUTHID=*authorization ID*

Details

If you specify a value for the AUTHID= option, the table name is qualified as *authid.tablename* before any SQL code is passed to DB2. If AUTHID= is not specified, the table name is not qualified before it is passed to DB2, and DB2 uses your OS/390 user ID as the qualifier. If you specify AUTHID= in a SAS/SHARE LIBNAME, the ID of the active server is the default ID.

AUTHID= is limited to 8 characters. It can also be specified with the SCHEMA alias.

DBNULL=

Indicates whether or not NULL is a valid value for the specified variables or columns when SAS creates a table and outputs data to DB2 tables.

Default value: NO

Syntax

DBNULL= (*variable-name–1*=YES | NO <...> *variable-name–n*=YES | NO)

Details

For a full description of this option, refer to Chapter 4, "SAS/ACCESS Data Set Options" .

DBTYPE=

Specifies whether or not to override the default DB2 data type(s) when SAS creates a table and outputs data to DB2 tables.

Default value: None

Syntax

DBTYPE= *(variable-name–1=DBMS-type <...> <variable-name-n=DBMS-type>)*

Details

By default, SAS numeric variables are assigned to FLOAT column types and character variables are assigned to VARCHAR column types. Use the DBTYPE= option to override these data type defaults. For a full description of this option, refer to Chapter 4, "SAS/ACCESS Data Set Options" .

IN=

Enables you to specify the database or tablespace in which you want to create a new table.

Default value: Default database or tablespace

Syntax

IN='*database-name.tablespace-name*'|'DATABASE *database-name*'

database-name.tablespace-name
specifies the names of the database and tablespace, which are separated by a period.

'DATABASE *database-name*'
specifies only the database name. In this case, you specify the word DATABASE, then a space and the database name. Enclose the entire specification in single quotes.

Details The IN= option is relevant only when you are creating a new table. If you omit this option, the default is to create the table in the default database or tablespace.

LOCATION=

Enables you to further qualify exactly where a table resides.

Default value: None

Syntax

LOCATION=*location*

Details

The location name maps to the location in the SYSIBM.SYSLOCATIONS catalog in the communication database.

In the DB2 engine, the location is converted to the first level of a three-level table name: LOCATION.AUTHID.TABLE. The connection to the remote DB2 subsystem is done implicitly by DB2 when DB2 receives a three-level name in an SQL statement.

If you specify LCOATION=, you must also specify the AUTHID= option.

READ_LOCK_TYPE=

Specifies whether or not a table lock is set on a DB2 table during a READ operation.

Default value: None

Syntax

READ_LOCK_TYPE= TABLE

Details

READ_LOCK_TYPE= is useful if you want to lock out concurrent changes in order to access an entire table as it is at that particular moment. You can also use it to prevent timeouts from contention with other application processes that are reading the same table.

DB2 locking is handled internally. No explicit locking is done. Consult your DBA to determine what the locking process is for your installation of DB2.

If you set READ_LOCK_TYPE=TABLE, then you must also set the CONNECTION= option to UNIQUE, which means that there is a separate physical connection for each table that is opened in your SAS application. You cannot share a connection when a DB2 table is locked. If you do not set the CONNECTION= option to UNIQUE, the SAS step fails.

If you set READ_LOCK_TYPE=TABLE, you might also want to evaluate whether or not you change the SPOOL= option from its default value of YES. If the table is locked and its data cannot be changed during the read transaction, you may not need to create a utility spool file.

For a full description of this option, refer to Chapter 4, "SAS/ACCESS Data Set Options".

See Also

UPDATE_LOCK_TYPE=

UPDATE_LOCK_TYPE=

Specifies whether or not a table lock is set on a DB2 table during an UPDATE operation.

Default value: None

Syntax

UPDATE_LOCK_TYPE= TABLE

Details

If you are updating a large part of a table, you can improve your performance by using UPDATE_LOCK_TYPE= to lock the entire table and prevent other application processes from having concurrent access to it. This is more efficient than locking each page as it is updated and unlocking it when the changes are committed. You can also use UPDATE_LOCK_TYPE= to prevent timeouts from contention with other application processes that are updating the same table.

DB2 locking is handled internally. Consult your DBA to determine what the locking process is for your installation of DB2.

If you set UPDATE_LOCK_TYPE=TABLE, then you must also set the CONNECTION= option to UNIQUE, which means that there is a separate physical connection for each table that is opened in your SAS application. You cannot share a connection when a DB2 table is locked. If you do not set the CONNECTION= option to UNIQUE, the SAS step fails.

If you set UPDATE_LOCK_TYPE=TABLE, you might also want to evaluate whether or not you want to change the SPOOL= option from its default value of YES. If the table is locked and its data cannot be changed during the update transaction, you might not need to create a utility spool file.

Note: Use UPDATE_LOCK_TYPE= with caution because it locks all the tables in a nonsegmented table space, even if they are not the table that is specifically locked. The locks are held until a commit point or until the connection is freed. All other application processes are locked out of the non-segmented table space for the duration of the lock. △

For a full description of this option, refer to Chapter 4, "SAS/ACCESS Data Set Options" .

See Also

READ_LOCK_TYPE=

SAS System Options and Settings for DB2

You can use the following SAS system options when you invoke a SAS session that accesses DB2:

DB2DBUG
is used to debug SAS code. When you submit a SAS statement that accesses DB2 data, DB2DBUG displays any DB2 SQL queries (generated by SAS) that are processed by DB2. The queries are written to the SAS log.

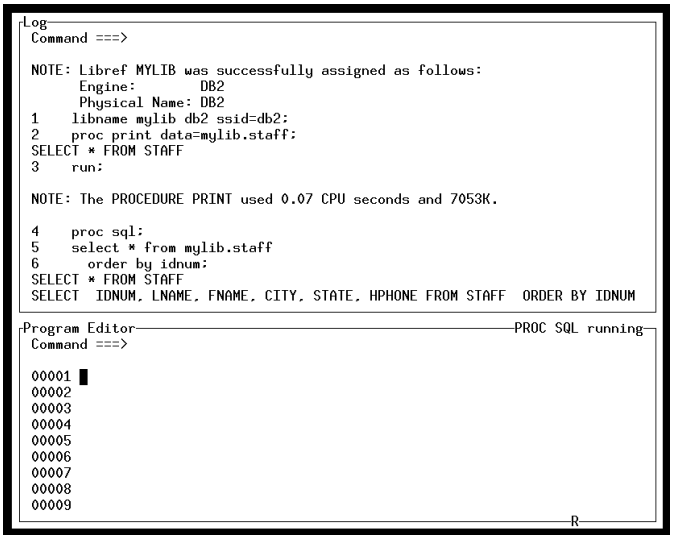
For example, if you submit a PROC PRINT statement that references a DB2 table, the DB2 SQL query is displayed in the SAS log. The SAS/ACCESS engine for DB2 generates the DB2 SQL query, as shown in Display 1.1 on page 12.

```
libname mylib db2 ssid=db2;

proc print data=mylib.staff;
run;

proc sql;
select * from mylib.staff
   order by idnum;
quit;
```

Display 1.1 DB2DBUG Display of DB2 SQL Queries



DB2 statements that appear in the SAS log are prepared and described in order to determine whether the DB2 table exists and can be accessed.

DB2DECPT=*decpoint-value*
specifies the setting of the DB2 DECPOINT option. The *decpoint-value* argument can be a . (period) or a , (comma). The default is a . (period).

DB2DECPT= is valid as part of the configuration file when you invoke the SAS System.

DB2IN='*database-name.tablespace-name*' | 'DATABASE *database-name*'
enables you to specify the database and tablespace in which you want to create a new table. The IN= option is relevant only when you are creating a new table. If you omit this option, the default is to create the table in the default database and tablespace.

database.tablespace specifies the names of the database and tablespace.

'DATABASE *database-name*' specifies only the database name. Enclose the entire specification in single quotes.

You can override the DB2IN= system option with the IN= libname or data set option.

DB2PLAN=*plan-name*
specifies the name of the plan that is used when connecting (or binding) SAS to DB2. SAS provides and supports this plan, which can be adapted for each user's site. The value for DB2PLAN= can be changed at any time during a SAS session, so that different plans can be used for different SAS steps. However, if you use more than one plan during a single SAS session, you must understand how and when the SAS/ACCESS engine for DB2 makes the connections. If one plan is in effect and you specify a new plan, the new plan does not affect the existing DB2 connections.

For details about how connections are managed by the SAS/ACCESS engine for DB2, see "Information for the Database Administrator" on page 34 and "Maximizing DB2 Performance" on page 31.

DB2RRS | NODB2RRS
specifies the attachment facility to be used for this SAS session when connecting to DB2. This option is an invocation only option.

Specify NODB2RRS, the default, to use the Call Attachment Facility (CAF). Specify DB2RRS to use the Recoverable Resource Manager Services Attachment Facility (RRSAF). For details about using RRSAF, see "Recoverable Resource Management Services Attachment Facility (RRSAF)" on page 36.

DB2SSID=*subsystem-name*
specifies the DB2 subsystem name. The *subsystem-name* argument is one to four characters that consist of letters, numbers, or national characters (#, $, or @); the first character must be a letter. The default value is DB2.

DB2SSID= is valid in the OPTIONS statement, as part of the configuration file, and when you invoke the SAS System.

You can override the DB2SSID= system option with the SSID= LIBNAME option.

DB2UPD=Y | N
specifies whether the user has privileges through the SAS/ACCESS interface to update DB2 tables. This option applies only to the user's updating privileges through the interface and not necessarily to the user's privileges while using DB2 directly. Altering the setting of DB2UPD= has no effect on your DBMS privileges, which have been set with the GRANT statement. The default is Y (Yes).

DB2UPD= is valid in the OPTIONS statement, as part of the configuration file, and when you invoke the SAS System.

Setting Your DB2 Subsystem Identifier

To connect to DB2, a valid DB2 subsystem name must be specified in one of the following ways:

- the DB2SSID= system option. This value is used by the SAS/ACCESS interface if no DB2 subsystem is specified. Refer to "SAS System Options and Settings for DB2" on page 12 for more information.
- the SSID= option in the PROC ACCESS statement.
- the SSID= statement of PROC DBLOAD.
- the SSID= option in the PROC SQL CONNECT statement, which is part of the Pass-Through facility.
- the SSID= option in the LIBNAME statement

If a site does not specify a valid DB2 subsystem when accessing DB2, the following message is generated:

```
ERROR: Cannot connect to DB2 subsystem XXXX,
  rc=12, reason code = 00F30006. Refer
  to the Call Attachment Facility documentation
  for an explanation.
```

where *XXXX* is the name of the subsystem to which SAS tried to connect. To find the correct value for your DB2 subsystem ID, contact your database administrator.

Capturing DB2 Return Codes Using SYSDBRC

Use the automatic macro variable SYSDBRC to capture DB2 return codes when using the DB2 engine. The macro variable is set to the last DB2 return code that was encountered only when execution takes place through the SAS/ACCESS interface to the DB2 engine. If you reference SYSDBRC before engine processing takes place, you receive this message:

```
WARNING: Apparent symbolic reference SYSDBRC
  not resolved.
```

Use SYSDBRC for conditional post-processing. Below is an example of how to abend a job. The table DB2TEST is dropped from DB2 after the view descriptor is created, resulting in a -204 code.

```
data test;
x=1;y=2;
proc dbload dbms=db2 data=test; table=db2test;
   in 'database test';
load;run;

proc access dbms=db2;
create work.temp.access;
table=user1.db2test;
create work.temp.view;
select all;
run;
proc sql;
execute(drop table db2test)by db2;
quit;
```

```
proc print data=temp;
run;

data _null_;
if "&sysdbrc" not in ('0','100') then
 do;
   put 'The DB2 Return Code is: ' "&sysdbrc";
   abort abend;
 end;
run;
```

Because the abend prevents the log from being captured, the SAS log can be captured by using the SAS system option ALTLOG. Refer to the *SAS Companion for the OS/390 Environment* for information on ALTLOG.

ACCESS Procedure: DB2 Specifics

Chapter 9, "ACCESS Procedure Reference" describes the generic options and procedure statements that enable you to create access descriptors, view descriptors, and SAS data files from DBMS data. The following section describes the DBMS-specific statements that you use in the SAS/ACCESS interface to DB2.

ACCESS Procedure Statements for DB2

To create an access descriptor, you use the DBMS=DB2 option and the TABLE= database-description statements in the PROC ACCESS step. This database-description statement supplies the DBMS name to the SAS System. The TABLE= statement must immediately follow the CREATE statement. The CREATE statement specifies the access descriptor to be created.

Database-description statements are required only when you create access descriptors. Because DB2 information is stored in an access descriptor, you do not need to repeat this information when you create view descriptors.

Note: Unlike some other SAS/ACCESS interface products, the SAS/ACCESS interface to DB2 does not use the following procedure statements: USER=, PASSWORD=, and DATABASE=. △

The SAS/ACCESS interface to DB2 uses the following procedure statements in interactive line, noninteractive, or batch mode.

PROC ACCESS <*access-descriptor-options*|*view-descriptor-options*>;
  CREATE *libref.member-name.* ACCESS|VIEW;
  UPDATE *libref.member-name.* ACCESS|VIEW;
  SSID=*DB2-subsystem-id*;
  SERVER=*DB2-database-system*|*DRDA-database-system*;
  LOCATION=*location*;
  TABLE=<*authorization-id.*>*table-name*;
  ASSIGN <=> YES|NO|Y|N;
  DROP <'>*column-identifier-1*<'><…<'>*column-identifier-n*<'>>;
  FORMAT<'>*column-identifier-1*<'><=>*SAS-format-name-1*
    <…<'>*column-identifier-n*<'><=>*SAS-format-name-n*>;
  QUIT ;

RENAME<'>*column-identifier-1*<'><=>*SAS-variable-name-1*
<...<'>*column-identifier-n*<'><=>*SAS-variable-name-n*;>

RESETALL|<'>*column-identifier-1*<'><...<'>*column-identifier-n*<'>>;

SELECTALL|<'>*column-identifier-1*<'><...<'>*column-identifier-n*<'>>;

SUBSET*selection-criteria*;

UNIQUE <=> YES|NO|Y|N;

LIST ALL | VIEW|<'>*column-identifier* <'>;

RUN ;

SERVER= *DRDA-database-system*;
enables direct access to DRDA resources (such as SQL/DS tables) from the SAS/ACCESS interface to DB2. SERVER= is an optional statement.

Enter a DRDA database system name assigned by your system administrator to make the connection to the desired database. Check with your system administrator for valid system names. You can connect with only one server at a time.

SSID=*DB2-subsystem-id*;
specifies the DB2 subsystem ID to use for the access descriptor. The *DB2-subsystem-id* is limited to four characters. Refer to "Setting Your DB2 Subsystem Identifier" on page 14 for more information on setting SSID=.

The SSID= statement is optional. If you omit it, the SAS System connects to the DB2 subsystem that is specified by the SAS system option DB2SSID=. If your site has not set DB2SSID=, the SSID= statement is required.

Consult your DBA to determine when the DRDA resources are set up properly. Refer to "Connections Using the Distributed Relational Database Architecture (DRDA)" on page 35 for more information.

LOCATION=*location*;
enables you to further qualify exactly where a table resides.

In the DB2 engine, the location is converted to the first level of a three-level table name: LOCATION.AUTHID.TABLE. The connection to the remote DB2 subsystem is done implicitly by DB2 when DB2 receives a three-level table name in an SQL statement.

LOCATION= is optional. If you omit it, SAS accesses the data from the local DB2 database.

TABLE= <*authorization-id.*>*table-name*;
identifies the DB2 table or DB2 view that you want to use to create an access descriptor. The *table-name* is limited to 18 characters. The TABLE= statement is required.

The *authorization-id* is a user ID or group ID that is associated with the DB2 table. The authorization ID is limited to eight characters. If you omit the authorization ID, DB2 uses your TSO (or OS/390) user ID. In batch mode, however, you must specify an authorization ID, otherwise an error message is generated.

DB2 Restriction on Connections

The DB2 interface engine restricts the maximum concurrent open cursors to 32 when working from a single connection. Note that if you are working with a SAS view that accesses other views, you could be opening more cursors than you are aware of.

Beginning in Version 7, there is no limit to the number of connections that you can have to DB2. The DB2 interface engine uses the Call Attachment Facility (CAF) or

Recoverable Resource Manager Service Attachment Facility (RRSAF) to make an explicit connection to the local DB2 subsystem. For each connection to the CAF, the DB2 interface engine attaches a separate OS/390 subtask. Note that if you establish too many separate connections, you can adversely affect your performance.

Examples: Creating Access Descriptors and View Descriptors

The following example creates an access descriptor and a view descriptor that are based on DB2 data.

```
options linesize=80;
libname adlib 'SAS-data-library';
libname vlib 'SAS-data-library';

proc access dbms=db2;

   /* create access descriptor  */
   create adlib.customr.access;
   table=testid.customers;
   ssid=db2;
   assign=yes;
   rename customer = custnum;
   format firstorder date7.;
   list all;

   /* create vlib.usacust view */
   create vlib.usacust.view;
   select customer state zipcode name
          firstorder;
   subset where customer like '1%';
run;
```

The next example uses the SERVER= statement to access the SQL/DS table TESTID.ORDERS from a remote location. Access and view descriptors are then created, based on the table.

```
libname adlib 'SAS-data-library';
libname vlib 'SAS-data-library';

proc access dbms=db2;
  create adlib.customr.access;
  table=testid.orders;
  server=testserver;
  assign=yes;
  list all;

 create vlib.allord.view;
  select ordernum stocknum shipto dateorderd;
    subset where stocknum = 1279;
run;
```

DBLOAD Procedure: DB2 Specifics

Chapter 10, "DBLOAD Procedure Reference" describes the generic options and procedure statements that enable you to create a DBMS table and to insert data in it. The following section describes the DBMS-specific statements that you use in the SAS/ACCESS interface to DB2.

DBLOAD Procedure Statements for DB2

To create and load a DB2 table, the SAS/ACCESS interface to DB2 uses the following statements in interactive line, noninteractive, or batch mode.

Note: Unlike some other SAS/ACCESS interface products, the SAS/ACCESS interface to DB2 does not use the following procedure statements: USER= and PASSWORD=. △

PROC DBLOAD DBMS=DB2 <DATA=<*libref.*> *SAS-data-set*> <APPEND>;
IN*database.tablespace* |'DATABASE *database*';
SSID=*DB2-subsystem-id*;
SERVER=*DB2-database-system*|*DRDA-database-system*;
TABLE=<*authorization-id.*>*table-name*;
ACCDESC=<*libref.*>*access-descriptor*;
COMMIT=*commit-frequency*;
DELETE*variable-identifier-1* <*...variable-identifier-n*>;
ERRLIMIT=*error-limit*;
LABEL;
LIMIT=*load-limit*;
NULLS *variable-identifier-1*= Y|N|D <*...variable-identifier-n*= Y|N|D>;
QUIT;
RENAME*variable-identifier-1*=<'> *column-name-1* <'>
<*...variable-identifier-n*= <'>*column-name-n* <'>>;
RESET ALL|*variable-identifier-1*<*...variable-identifier-n*>;
SQL *DB2 SQL-statement*;
TYPE *variable-identifier-1 = 'column-type-1'*
<*...variable-identifier-n = 'column-type-n'* >;
WHERE *SAS-where-expression*;
LIST <ALL|COLUMN|*variable-identifier*>;
LOAD;

RUN;

IN *database.tablespace*|'DATABASE *database*';
specifies the nasme of the database or the table space in which you want to store the new DB2 table. A table space can contain multiple tables. The *database* and *tablespace* arguments are each limited to 18 characters. The IN statement must immediately follow the PROC DBLOAD statement.

database.tablespace
specifies the names of the database and the table space, which are separated by a period.

'DATABASE *database* '
specifies only the database name. In this case, you specify the word **DATABASE**, then a space and the database name. Enclose the entire specification in single quotes.

NULLS *variable-identifier-1* =Y|N|D < . . . *variable-identifier-n* =Y|N|D >;
enables you to specify whether the DB2 columns that are associated with the listed SAS variables allow NULL values. By default, all columns accept NULL values. Refer to Chapter 10, "DBLOAD Procedure Reference" for more information.

The NULLS statement accepts any one of these three values:

Y — specifies that the column accepts NULL values. This is the default.

N — specifies that the column does not accept NULL values.

D — specifies that the column is defined as NOT NULL WITH DEFAULT.

Refer to "DB2 NULLs and DB2 Default Values" on page 28 for DB2-specific information on NULL values.

SSID=*DB2-subsystem-id*;
specifies the DB2 subsystem ID to use for the access descriptor. The *DB2-subsystem-id* is limited to four characters. Refer to "Setting Your DB2 Subsystem Identifier" on page 14 for more information on setting SSID= or contact your DBA.

The SSID= statement is optional. If you omit it, the SAS System connects to the default DB2 subsystem that is specified by the SAS system option DB2SSID=. If your site has not set DB2SSID=, the SSID= statement is required.

SERVER=*DRDA-database-system*;
enables direct access to DRDA resources (such as SQL/DS tables) from the SAS/ACCESS interface to DB2. SERVER= is an optional statement.

Enter a DRDA database system name that is assigned by your system administrator to make the connection to the desired database system. Check with your system administrator for valid database system names. You can connect with only one server at a time.

TABLE= <*authorization-id.*>*table-name*;
identifies the DB2 table that you want to use to create. The *table-name* is limited to 18 characters. A DB2 table by this name cannot already exist, unless you are using the APPEND option on the PROC DBLOAD statement. The TABLE= statement is required.

The *authorization-id* is a user ID or group ID that is associated with the DB2 table. The authorization ID is limited to eight characters. If you omit the authorization ID, DB2 uses your TSO (or OS/390) user ID except in batch mode; in batch mode, you must specify an authorization ID.

The following example creates a new DB2 table, TESTID.INVOICE, from the DLIB.INVOICE data file. The AMTBILLED column and the 5th column in the table (AMOUNTINUS) are renamed. You must be granted the appropriate privileges in order to create new DB2 tables.

```
libname adlib 'SAS-data-library';
libname dlib 'SAS-data-library';

proc dbload dbms=db2 data=dlib.invoice;
   ssid=db2;
   table=testid.invoice;
```

```
   accdesc=adlib.invoice;
   rename amtbilled = amountbilled
          5 = amountindollars;
   nulls invoicenum=n amtbilled=n;
   load;
run;
```

Suppose that you just created a SAS data set, WORK.SCHEDULE, which includes the names and work hours of your employees. You can use the SERVER= command to create the DB2 table TESTID.SCHEDULE and load it with the schedule data on the DRDA resource, TESTSERVER, as shown in the next example.

```
libname adlib 'SAS-data-library';

proc dbload dbms=db2 data=work.schedule;
  in sample;
  server=testserver;
  accdesc=adlib.schedule;
  table=testid.schedule;
  list all;
  load;
run;
```

SQL Procedure Pass-Through Facility: DB2 Specifics

Chapter 6, "SQL Procedure's Interaction with SAS/ACCESS Software" describes the generic PROC SQL statements that you use to connect to and disconnect from a DBMS, to send DBMS-specific statements to the DBMS, and to retrieve DBMS data for your SAS programs. The following section describes the DBMS-specific arguments that you use in the CONNECT statement"Arguments to Connect to DB2" on page 20.

Arguments to Connect to DB2

The CONNECT statement is optional when you are connecting to DB2. DB2 has two *database-connection-arguments* that you can specify in this statement. CONNECT can also be used to connect to multiple DB2 systems.

CONNECT TO DB2 <AS *alias*> <(SSID=*DB2-subsystem-id*>
<SERVER=*DRDA-database-system*)>;

SSID=*DB2-subsystem-id*
specifies the DB2 subsystem ID that you want to connect to. The ID is limited to four characters.

The SSID= argument is optional. If you omit it, the SAS System connects to the default DB2 subsystem that is specified by the SAS system option DB2SSID=. If your site has not set DB2SSID=, the SSID= argument is required.

Refer to "Setting Your DB2 Subsystem Identifier" on page 14 for information on setting your subsystem ID or contact your DBA.

SERVER=*DRDA-database-system*
enables direct access to DRDA resources (such as SQL/DS tables) from the SAS/ACCESS interface to DB2. SERVER= is an optional statement.

Enter a DRDA database system name that is assigned by your system administrator to make the connection to the desired database system. Check with your system administrator for valid database system names. You can connect with only one system at a time.

The following example connects to DB2 and sends it two EXECUTE statements to process:

```
proc sql;
   connect to db2 (ssid=db2);
   execute (create view testid.whotookorders as
         select ordernum, takenby, firstname,
              lastname, phone
            from testid.orders, testid.employees
            where testid.orders.takenby=
                  testid.employees.empid)
           by db2;
   execute (grant select on testid.whotookorders
            to testuser) by db2;
   disconnect from db2;
quit;
```

The following example omits the optional CONNECT statement, uses the default setting for SSID=, and performs a query shown in italics on the TESTID.CUSTOMERS table:

```
proc sql;
   select *
      from connection to db2
         (select * from testid.customers
             where customer like '1%');
   disconnect from db2;
quit;
```

The next example creates the PROC SQL view VLIB.STOCKORD that is based on the table TESTID.ORDERS. The table TESTID.ORDERS is an SQL/DS table that is accessed through DRDA.

```
libname vlib 'SAS-data-library'

proc sql;
  connect to db2 (server=testserver);
  create view vlib.stockord as
     select * from connection to
        db2(select ordernum, stocknum,
            shipto, dateorderd
            from testid.orders);
  disconnect from db2;
quit;
```

The DB2UTIL Procedure

You can use the DB2UTIL procedure to insert, update, or delete rows in a DB2 table using data from a SAS data set. You can choose one of two methods of processing: creating an SQL output file or executing directly. PROC DB2UTIL runs interactively, noninteractively, or in batch mode.

Note: The DB2UTIL procedure is supported in order to provide compatibility with Version 5 of the SAS/ACCESS interface to DB2. It will not be added to other SAS/ACCESS DBMS interfaces, nor will the enhancement of this procedure for future releases of SAS/ACCESS be guaranteed. It is recommended that new applications be written by using the new LIBNAME features. △

The DB2UTIL procedure uses the data in an input SAS data set, along with your mapping specifications, to generate SQL statements that modify the DB2 table. The DB2UTIL procedure can perform the following:

DELETE	deletes rows from the DB2 table according to the search condition that you specify.
INSERT	builds rows for the DB2 table from the SAS observations, according to the map that you specify, and inserts the rows.
UPDATE	sets new column values in your DB2 table by using the SAS variable values that are indicated in your map.

When you execute the DB2UTIL procedure, you specify an input SAS data set, an output DB2 table, and how to modify the data. To generate data, you must also supply instructions for mapping the input SAS variable values to the appropriate DB2 columns.

In each execution, the procedure can generate and execute SQL statements to perform one type of modification only. However, you can also supply your own SQL statements (except the SQL SELECT statement) to perform various modifications against your DB2 tables, and the procedure will execute them.

Refer to “Modifying DB2 Data” on page 24 for more information on the types of modifications that are available and how they are used. Refer to “DB2UTIL Example” on page 25 for an example of using DB2UTIL.

DB2UTIL Statements and Options

The PROC DB2UTIL statement invokes the DB2UTIL procedure. The following statements are used with PROC DB2UTIL:

PROC DB2UTIL <*options*>;
MAPTO *SAS-name-1*=DB2-name-1 <...SAS-name-n=DB2-name-n>;
RESETALL|*SAS-name*| COLS;
SQL *SQL-statement*;
UPDATE;
WHERE *SQL-WHERE-clause*;
ERRLIMIT=*error-limit*;
EXIT;

Options:

DATA=*SAS-data-set* | <*libref.*>*SAS-data-set*
specifies the name of the SAS data set that contains the data with which you want to update the DB2 table. DATA= is required unless you specify an SQL file with the SQLIN= option.

TABLE=*DB2-tablename*
specifies the name of the DB2 table that you want to update. TABLE= is required unless you specify an SQL file with the SQLIN= option.

FUNCTION= D | I | U | DELETE | INSERT | UPDATE
specifies the type of modification to perform on the DB2 table by using the SAS data set as input. Refer to "Modifying DB2 Data" on page 24 for a detailed description of this option. FUNCTION= is required unless you specify an SQL file with the SQLIN= option.

You can also specify these options with PROC DB2UTIL:

COMMIT=*number*
specifies the maximum number of SQL statements to execute before issuing an SQL COMMIT statement to establish a syncpoint. The default is 3.

ERROR=*fileref* |*fileref.member*
specifies an external file where error information is logged. When DB2 issues an error return code, the procedure writes all relevant information, including the SQL statement that is involved, to this external file. If you omit the ERROR= statement, the procedure writes the error information to the SAS log.

LIMIT=*number*
specifies the maximum number of SQL statements to issue in an execution of the procedure. The default value is 5000. If you specify LIMIT=0, no limit is set. The procedure processes the entire data set regardless of its size.

SQLIN=*fileref* | *fileref.member*
specifies an intermediate SQL output file that is created by a prior execution of PROC DB2UTIL by using the SQLOUT= option. The file that is specified by SQLIN= contains SQL statements to update a DB2 table. If you specify an SQLIN= file, then the procedure reads the SQL statements and executes them in line mode. When you specify an SQLIN= file, DATA=, TABLE=, and SQLOUT= are ignored.

SQLOUT=*fileref* | *fileref.member*
specifies an external file where the generated SQL statements are to be written. This file is either an OS/390 sequential data set or a member of an OS/390 partitioned data set. Use this option to update or delete data.

When you specify the SQLOUT= option, the procedure edits your specifications, generates the SQL statements to perform the update, and writes them to the external file for later execution. When they are input to the later run for execution, the procedure passes them to DB2.

SSID=*subsystem-name*
specifies the name of the DB2 subsystem that you want to access. If you omit DB2SSID=, the subsystem name defaults to DB2.

MAPTO Statement

MAPTO *SAS-name-1=DB2-name-1<... SAS-name-n=DB2-name-n>*;

The MAPTO statement maps the SAS variable name to the DB2 column name. You can specify as many values in one MAPTO statement as you want.

RESET Statement

RESET ALL | *SAS-name* | COLS;

Use the RESET statement to erase the editing that was done to SAS variables or DB2 columns. The RESET statement can perform one or more of the following actions:

ALL	resets all previously entered map and column names to the procedure's default values.
SAS-name	resets the map entry for that SAS variable.
COLS	resets the altered column values.

SQL Statement

SQL *SQL-statement*;

The SQL statement specifies an SQL statement that you want the procedure to execute dynamically. The procedure rejects SQL SELECT statements.

UPDATE Statement

UPDATE;

The UPDATE statement causes the table to be updated by using the mapping specifications that you supply. If you do not specify an input or an output mapping data set or an SQL output file, the table is updated by default.

If you have specified an output mapping data set in the SQLOUT=option, PROC DB2UTIL creates the mapping data set and ends the procedure. However, if you specify UPDATE, the procedure creates the mapping data set and updates the DB2 table.

WHERE Statement

WHERE *SQL-WHERE-clause*;

The WHERE statement specifies the SQL WHERE clause that you want to use in the update of the DB2 table. This statement is combined with the SQL statement generated from your mapping specifications. Any SAS variable names in the WHERE clause are substituted at that time. For example:

```
where db2col = %sasvar;
```

ERRLIMIT Statement

ERRLIMIT=*error-limit*;

The ERRLIMIT statement specifies the number of DB2 errors that are permitted before the procedure terminates.

EXIT Statement

EXIT;

The EXIT statement exits from the procedure without further processing. NO output data is written, and no SQL statements are issued.

Modifying DB2 Data

The DB2UTIL procedure generates SQL statements by using data from an input SAS data set. However, the SAS data set plays a different role for each type of modification that is available through PROC DB2UTIL. The following sections show how you use each type and how each type uses the SAS data set to make a change in the DB2 table.

Inserting Data

INSERT enables you to insert observations from a SAS data set into a DB2 table as rows in the table. To use the INSERT function, name the SAS data set that contains

the data you want to insert and the DB2 table to which you want to add information in the PROC DB2UTIL statement. You can then use the MAPTO statement to map values from SAS variables to columns in the DB2 table. If you do not want to insert the values for all the variables in the SAS data set into the DB2 table, map only the variables that you want to insert.

Updating Data

UPDATE enables you to change the values in DB2 table columns by replacing them with values from a SAS data set. You can change a column value to another value for every row in the table, or you can change column values only when certain criteria are met. For example, you can change the value of the DB2 column NUM to 10 for every row in the table. You can also change the value of the DB2 column NUM to the value in the SAS variable NUMBER, providing that the value of the DB2 column NAME and the SAS data set variable NAME match.

You specify the name of the SAS data set and the DB2 table to be updated when you execute PROC DB2UTIL. You can specify that only certain variables be updated by naming only those variables in your mapping specifications.

You can use the WHERE clause to specify that only the rows on the DB2 table that meet certain criteria are updated. For example, you can use the WHERE clause to specify that only the rows with a certain range of values be updated, or you can specify that rows will be updated when a certain column value in the row matches a certain SAS variable value in the SAS data set. In this case, you could have a SAS data set with several observations in it. For each observation in the data set, the DB2UTIL procedure updates the values for all rows in the DB2 table that have a matching value. Then the procedure goes on to the next observation in the SAS data set and continues to update values in DB2 columns in rows that meet the comparison criteria.

Deleting Data

DELETE enables you to remove rows from a DB2 table when a certain condition is met. You can delete rows from the table when a DB2 column value in the table matches a SAS variable value in the SAS data set. Name the DB2 table from which you want to delete rows and the SAS data set that contains the target deletion values in the PROC DB2UTIL statement. Then use the WHERE statement to specify the DB2 column name and the SAS variable whose values must match before the deletion is performed.

If you want to delete values that are based on criteria other than values in SAS data variables (for example, deleting every row with a department number of 600), then you can use an SQL DELETE statement in an SQL statement.

DB2UTIL Example

The following example uses DB2UTIL's UPDATE function to update a list of telephone extensions from a SAS data set. The master list of extensions is in the DB2 table TESTID.EMPLOYEES and will be updated from SAS data set TRANS. First you must create the SAS data set:

```
options db2dbug;

data trans;
empno=321783;ext='3999';
output;
empno=320001;ext='4321';
output;
empno=212916;ext='1300';
```

```
output;
run;
```

Next, specify the data set in PROC DB2UTIL.

```
proc db2util data=trans table=testid.employees function=u;
   mapto ext=phone;
   where empid=%empno;
   update; run;
```

The row that includes EMPID=320001 was not found in the TESTID.EMPLOYEES table and therefore was not updated. The warning that appears in the SAS log can be ignored.

DB2 Naming Conventions

DB2 objects that can be named include tables, views, columns, and indexes. Use the following DB2 naming conventions:

- A name must start with a letter. If the name is in quotes, it can start with and contain any character. Depending on how your string delimiter is set, quoted strings can contain quotes such as "O'Malley".
- The following objects can have names from 1 to 18 characters long: a column, cursor, index, table, view, alias, synonym, collection ID, statement name, or correlation name.

  The following objects can have names from 1 to 8 characters long: authorization ID, referential constraint, database, table space, storage group, package, or plan.

  A location name can be 1 to 16 characters long.
- A name can contain the letters A through Z, the digits 0 through 9, and national characters (#, $, or @).
- A name is not case-sensitive. For example, CUSTOMER is the same as customer. However, if the name of the object is in quotes, it is case-sensitive.
- The name cannot be a DB2 reserved word.
- A name cannot be the same as another DB2 object. For example, each column name within the same table must be unique.

DB2 Data Types

Every column in a table has a name and a data type. DB2 data types fall into three categories: types for string data; types for numeric data; and types for dates, times, and timestamps. The categories, followed by the data types within each category, are listed in the following sections. The SAS/ACCESS interface to DB2 handles all DB2 data types. This section describes how the DB2 engine treats each of these data types.

String Data

The DB2 string data types are listed here.

CHAR(*n*)
: specifies a fixed-length column of length *n* for character string data. The maximum for *n* is 254.

VARCHAR(*n*)
specifies a varying-length column for character string data. *n* specifies the maximum length of the string. If *n* is greater than 254, the column is a long string column. DB2 imposes some restrictions on referencing long string columns.

LONG VARCHAR
specifies a varying-length column for character string data. DB2 determines the maximum length of this column. A column defined as LONG VARCHAR is always a long string column and, therefore, subject to referencing restrictions.

GRAPHIC(*n*), VARGRAPHIC(*n*), LONG VARGRAPHIC
specifies graphic strings and is comparable to the types for character strings. However, *n* specifies the number of double-byte characters, so the maximum value for *n* is 127. If *n* is greater than 127, the column is a long string column and is subject to referencing restrictions.

Numeric Data

The DB2 numeric data types are listed here.

SMALLINT
specifies a small integer. Values in a column of this type can range from –32,768 through +32,767.

INTEGER | INT
specifies a large integer. Values in a column of this type can range from –2,147,483,648 through +2,147,483,647.

REAL | FLOAT(*n*)
specifies a single-precision, floating-point number. If *n* is omitted or if *n* is greater than 21, the column is double-precision. Values in a column of this type can range from approximately –7.2E+75 through 7.2E+75.

FLOAT(*n*) | DOUBLE PRECISION | FLOAT | DOUBLE
specifies a double-precision, floating-point number. *n* can range from 22 through 53. If *n* is omitted, 53 is the default. Values in a column of this type can range from approximately –7.2E+75 through 7.2E+75.

DECIMAL(*p*,*s*) | DEC(*p*,*s*)
specifies a packed-decimal number. *p* is the total number of digits (precision) and *s* is the number of digits to the right of the decimal point (scale). The maximum precision is 31 digits. The range of *s* is $0 \le s \le p$.

If *s* is omitted, 0 is assigned and *p* may also be omitted. Omitting both *s* and *p* results in the default DEC(5,0). The maximum range of *p* is $1 - 10^{31}$ to $10^{31} - 1$.

Even though the DB2 numeric columns have these distinct data types, the DB2 engine accesses, inserts, and loads all numerics as FLOATs.

Dates, Times, and Timestamps

DB2 date and time data types are similar to SAS date and time values in that they are stored internally as numeric values and are displayed in a site-chosen format. The DB2 data types for dates, times, and timestamps are listed here. Note that columns of these data types may contain data values that are out of range for the SAS System, which handles dates from 1582 A.D. through 20,000 A.D.

DATE
specifies date values in the format YYYY-MM-DD. For example, January 25, 1989, is input as 1989-01-25. Values in a column of this type can range from 0001-01-01 through 9999-12-31.

TIME
specifies time values in the format HH.MM.SS. For example, 2:25 p.m. is input as 14.25.00. Values in a column of this type can range from 00.00.00 through 24.00.00.

TIMESTAMP
combines a date and time and adds a microsecond to make a seven-part value of the format YYYY-MM-DD-HH.MM.SS.MMMMMM. For example, a timestamp for precisely 2:25 p.m. on January 25, 1989, is 1989-01-25-14.25.00.000000. Values in a column of this type can range from 0001-01-01-00.00.00.000000 through 9999-12-31-24.00.00.000000.

DB2 NULLs and DB2 Default Values

DB2 has a special value that is called *NULL*. This value means an absence of information. It is analogous to the SAS System's missing value.

Columns can be defined so that they do not allow NULL data. NOT NULL would indicate, for example, that DB2 does not allow a row to be added to the TESTID.CUSTOMERS table unless there's a value for CUSTOMER.

Columns can also be defined as NOT NULL WITH DEFAULT. The following table lists the default values assigned by DB2 to columns that are defined as NOT NULL WITH DEFAULT. An example of such a column is STATE in TESTID.CUSTOMERS. If a column is omitted from a view descriptor, default values are assigned to the column. However, if a column is specified in a view descriptor and it has no values, no default values are assigned.

Table 1.1 Default Values Assigned by DB2 for columns defined as NOT NULL WITH DEFAULT

DB2 Column Type	DB2 Default*
CHAR(*n*) | GRAPHIC(*n*)	blanks, unless the NULLCHARVAL= option is specified
VARCHAR | LONG VARCHAR | VARGRAPHIC | LONG VARGRAPHIC	empty string
SMALLINT | INT | FLOAT | DECIMAL | REAL	0
DATE	current date, derived from the system clock
TIME	current time, derived from the system clock
TIMESTAMP	current timestamp, derived from the system clock

* The default values that are listed in this table pertain to values that are assigned by DB2.

Knowing whether a DB2 column allows NULL values or whether DB2 supplies a default value can assist you in writing selection criteria and in entering values to update a table. Unless a column is defined as NOT NULL or NOT NULL WITH DEFAULT, the column allows NULL values.

LIBNAME Statement Data Conversions

The following table shows the default SAS System variable formats that the DB2 engine assigns to DB2 data types during input operations.

Table 1.2 LIBNAME Statement: Default SAS Formats for DB2 Data Types

DB2 Column Type	Default SAS Format
CHAR(*n*)	$*n*. (*n*<=254)
VARCHAR(*n*)	$*n*.
	$255. (*n*>255)
LONG VARCHAR	$*n*.
GRAPHIC(*n*), VARGRAPHIC(*n*), LONG VARGRAPHIC	$*n*.(*n*<=127) $127. (*n*>127)
INTEGER	*m*.*n*
SMALLINT	*m*.*n*
DECIMAL(*m*,*n*)	*m*.*n*
FLOAT	none
NUMERIC(*m*,*n*)	*m*.*n*
DATE	DATE9.
TIME	TIME8.
DATETIME	DATETIME30.6

The following table shows the default DB2 data types that are assigned to SAS variable formats during output operations.

Table 1.3 LIBNAME Statement: Default DB2 Data Types for SAS Variable Formats

SAS Variable Format	DB2 Data Type
$*w*., $CHAR*w*., $VARYING*w*., $HEX*w*.	CHARACTER
any date format	DATE
any time format	TIME
any datetime format	TIMESTAMP
all other numeric formats	FLOAT

ACCESS Procedure Data Conversions

The following table shows the default SAS System variable formats that the ACCESS procedure assigns to DB2 data types.

Table 1.4 ACCESS Procedure: Default SAS Formats for DB2 Data Types

DB2 Column Type	Default SAS Format
CHAR(*n*)	$*n*. (*n*<=199)
VARCHAR(*n*)	$*n*.
	$200. (*n*>200)
LONG VARCHAR	$*n*.
GRAPHIC(*n*), VARGRAPHIC(*n*), LONG VARGRAPHIC	$*n*.(*n*<=127)
	$127. (*n*>127)
INTEGER	11.0
SMALLINT	6.0
DECIMAL(*m*,*n*)	*m*+2.*s*
	for example, DEC(6,4) = 8.4
REAL	E12.6
DOUBLE PRECISION	E12.6
FLOAT(*n*)	E12.6
FLOAT	E12.6
NUMERIC(*m*,*n*)	*m*.*n*
DATE	DATE7.
TIME	TIME8.
DATETIME	DATETIME30.6

Note: You can use the YEARCUTOFF= option to make your DATE7. dates comply with Year 2000 standards. For more information about this SAS system option, see *SAS Language Reference: Dictionary*. △

DBLOAD Procedure Data Conversions

The following table shows the default DB2 data types that the DBLOAD procedure assigns to SAS variable formats.

Table 1.5 DBLOAD Procedure: Default DB2 Data Types for SAS Variable Formats

SAS Variable Format	DB2 Data Type
$*w*., $CHAR*w*., $VARYING*w*., $HEX*w*.	CHARACTER
any date format	DATE
any time format	TIME

SAS Variable Format	DB2 Data Type
any datetime format	TIMESTAMP
all other numeric formats	FLOAT

Maximizing DB2 Performance

Among the factors that affect DB2 performance are the size of the table that is being accessed and the form of the SQL SELECT statement. If the table that is being accessed is larger than 10,000 rows (or 1,000 pages), you should evaluate all SAS programs that access the table directly. When you evaluate the programs, consider the following questions:

- Does the program need all of the columns that the SELECT statement retrieves?
- Do the WHERE clause criteria retrieve only those rows that are needed for subsequent analysis?
- Is the data going to be used by more than one procedure in one SAS session? If so, consider extracting the data into a SAS data file for SAS procedures to use instead of allowing the data to be accessed directly by each procedure.
- Do the rows need to be in a particular order? If so, can an indexed column be used to order them? If there is no index column, is DB2 doing the sort?
- Do the WHERE clause criteria allow DB2 to use the available indexes efficiently?
- What kind of locks does DB2 need to acquire?
- Are the joins being passed to DB2?
- Can your DB2 system use parallel processing to access the data more quickly?

DB2 has a Resource Limit Facility to limit the execution time of dynamic SQL statements. If the time limit is exceeded, the dynamic statement is terminated and the SQL code -905 is returned. The following list describes several situations in which the RLF could stop a user from consuming large quantities of CPU time:

- An extensive join of DB2 tables with the SAS System SQL procedure.
- An extensive search by the FSEDIT, FSVIEW, or FSBROWSE procedures or an SCL application.
- Any extensive extraction of data from DB2.
- An extensive select.
- An extensive load into a DB2 table. In this case, you can break up the load by lowering the commit frequency.

There are several things that you can do in your SAS application to make the DB2 engine perform better:

- Set the SAS system option DB2DBUG. This option prints the dynamic SQL that is generated by the DB2 engine and all other SQL that is executed by the DB2 engine to the SAS log. You can then verify that all WHERE clauses, PROC SQL joins, and ORDER BY clauses are being passed to DB2. This option is for debugging purposes and should not be set once the SAS application is used in production. The NODB2DBUG option deactivates this behavior.
- Verify that all SAS procedures and DATA step code that read DB2 data share connections where possible. You can do this by using one libref to reference all of

the SAS applications that read DB2 data and by accepting the default value of SHAREDREAD for the CONNECTION= option.

- If your DB2 subsystem supports parallel processing, you can assign a value to the CURRENT DEGREE special register. Setting this register may enable your SQL query to use parallel operations. You can set the special register by using the LIBNAME options DBCONINIT= or DBLIBINIT= with the SET statement as shown in the following example:

```
libname mydb2 db2 dbconinit="SET CURRENT DEGREE='ANY'";
```

- Use the view descriptor WHERE clause or the DBCONDITION= option to pass WHERE clauses to DB2. You can also use these methods to pass sort operations to DB2 with the ORDER BY clause instead of performing a sort within SAS.
- If you are using a SAS application or an SCL application that reads the DB2 data twice, let the DB2 engine spool the DB2 data. This happens by default because the default value for the SPOOL= option is YES.

  The spool file is read both when the application rereads the DB2 data and when the application scrolls forward or backward through the data. If you do not use spooling, and you need to scroll backward through the DB2 table, the DB2 engine must start reading from the beginning of the data and read down to the row that you want to scroll back to.
- Use the SQL procedure to pass joins to DB2 instead of using the MATCH MERGE capability (that is, merging with a BY statement) of the DATA step.
- Use the DBKEY= option when you are doing SAS processing that involves the KEY= option. When you use the DBKEY= option, the DB2 engine generates a WHERE clause that uses parameter markers. During the execution of the application, the values for the key are substituted into the parameter markers in the WHERE clause.

  If you don't use the DBKEY= option, a new WHERE clause is generated for every key value. The SQL query with the new WHERE clause must be optimized in the DB2 PREPARE process every time the key value changes. The SQL query that uses the WHERE clause with parameter markers is optimized, or PREPAREd, only once.
- Consider using stored procedures when they can improve performance in client/server applications by reducing network traffic. You can execute a stored procedure by using the DBCONINIT= or DBLIBINIT= LIBNAME options.

Making the Most of Your Connections

Beginning in Version 7, the DB2 engine supports more than one connection to DB2 per SAS session. This is an improvement over Version 6 in a number of ways, especially in a server environment. One advantage is being able to segregate tasks that fetch rows from a cursor from tasks that must issue commits. This eliminates having to resynchronize the cursor, prepare the statement, and fetch rows until you are positioned back on the row you were on. This separation also allows tasks that must issue commits to eliminate locking contention to do so sooner, since they are not delayed until after cursors are closed to prevent having to resynchronize. In general, tables opened for input fetch from cursors and do not issue commits, while update opens may, and output opens do, issue commits.

You can control how the DB2 engine uses connections by using the CONNECTION= option on the LIBNAME statement. At one extreme is CONNECTION=UNIQUE, which causes each table access, whether it is for input, update, or output, to create and use its own connection. Conversely, CONNECTION=SHARED means that only one connection is made, and that input, update, and output accesses all share that connection.

The default value for the CONNECTION= option is CONNECTION=SHAREDREAD, which means that tables opened for input share one connection, while update and output opens get their own connections. CONNECTION=SHAREDREAD allows the best separation between tasks that fetch from cursors and tasks that must issue commits, eliminating the resynchronizing of cursors.

The values GLOBAL and GLOBALREAD perform similarly to SHARED and SHAREDREAD. The difference is that you can share the given connection across any of the librefs that you specify as GLOBAL or GLOBALREAD.

Although the default value of CONNECTION= SHAREDREAD is optimal, there are times when another value might be better. If you must use multiple librefs, you might want to set them each as GLOBALREAD. This way, you will have one connection for all of your input opens, regardless of which libref you use, as opposed to one connection per libref for input opens. In a single-user environment (as opposed to a server session), you might know that you will not have multiple opens occurring at the same time. In this case, you might want to use SHARED (or GLOBAL for multiple librefs). This eliminates the overhead of creating separate connections for input, update, and output transactions, while having only one open at a time eliminates the problem of having to resynchronize input cursors if a commit occurs.

Another reason for using SHARED or GLOBAL is the case of opening a table for output while opening another table within the same database for input. This can result in a -911 deadlock situation unless both opens occur in the same connection.

As explained in “Information for the Database Administrator” on page 34, the first connection to DB2 is made from the main SAS task. Subsequent connections are made from corresponding subtasks, which the DB2 engine attaches; DB2 allows only one connection per task. Due to the system overhead of intertask communication, the connection established from the main SAS task is a faster connection in terms of CPU time. Since this is true, if you are reading or writing large numbers of rows, you will have better performance (less CPU time) if you use the first connection for these operations. If you are only reading rows, SHAREDREAD or GLOBALREAD can share the first connection. However, if you are both reading and writing rows (input and output opens), you can use CONNECTION=UNIQUE to make each open use the first connection. UNIQUE causes each open to have its own connection. If you only have one open at a time, and some are input while others are output (for large amounts of data), the performance benefit of using the main SAS task connection far outweighs the overhead of establishing a new connection for each open.

One other type of connection that the DB2 engine uses, and which is not user controllable, is the utility connection. This connection is used to access the system catalog, issues commits to release locks, and is a separate connection. Utility procedures such as DATASETS and CONTENTS can cause this connection to be created, although other actions necessitate it as well. There is one connection of this type per libref, but it is not created until it is needed. If you have critical steps which must use the main SAS task connection for performance reasons, refrain from using the DEFER=YES option on the LIBNAME statement. It is possible that the utility connection can be established from that task, causing the connection you use for your opens to be from a slower subtask.

In summary, there is not one value for the CONNECTION= option which works best in all possible situations. You might need to try different values and arrange your SAS programs in different ways to obtain the best performance possible.

For additional information about

- DB2–specific LIBNAME options, refer to “SAS/ACCESS LIBNAME Options” on page 4.
- SAS system options, refer to “SAS System Options and Settings for DB2” on page 12

- SAS/ACCESS LIBNAME options, refer to Chapter 3, "SAS/ACCESS LIBNAME Statement" .
- SAS/ACCESS data set options, refer to Chapter 4, "SAS/ACCESS Data Set Options" .

Information for the Database Administrator

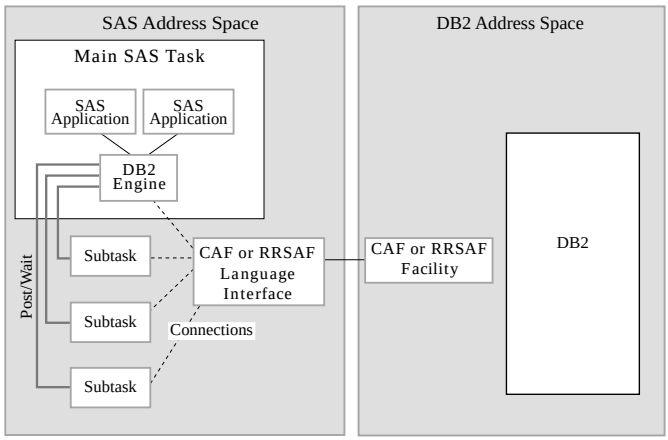
This section includes information about how the DB2 engine works, how SAS connects to DB2, and how the DB2 engine accesses DB2 system catalogs.

How the DB2 Engine Works

The DB2 engine uses the Call Attachment Facility (CAF) or the Recoverable Resource Management Services Attachment Facility (RRSAF) as the Application Programming Interface (API) in order to communicate to the local DB2. Both attachment facilities enable programs to connect to and use DB2 for SQL statements, or commands. The DB2 engine uses the attachment facilities to establish and control its connections to the local DB2 subsystem. DB2 allows only one connection for each task control block (TCB), or task. The SAS System and SAS executives run under one TCB, or task.

The design of the new, dynamic DB2 LIBNAME engine gives SAS users the ability to connect to DB2 more than once. Because the CAF and RRSAF allow only one connection per TCB, the DB2 engine attaches a subtask for each subsequent connection that is initiated. The DB2 engine uses the ATTACH, DETACH, POST, and WAIT assembler macros to create and communicate with the subtasks. The DB2 engine does not limit the number of connections/subtasks that a single SAS user can initiate. Display 1.2 on page 34 illustrates how the DB2 engine works.

Display 1.2 Design of the DB2 LIBNAME Engine



How and When Connections Are Made

The DB2 engine always makes an explicit connection to the local DB2 subsystem (SSID). When a connection executes successfully, a thread to DB2 is established. For each thread's or task's connection, DB2 establishes authorization identifiers (AUTHIDs).

The DB2 engine determines when to make a connection to DB2 based on the type of open mode that the SAS application requests for the DB2 tables. There are three distinct types of open modes that a SAS application can request: read, update, and output. The default behavior for the DB2 engine is to share the connection for all open modes of read for each DB2 LIBNAME statement. For every update and output open mode, the DB2 engine acquires a separate connection to DB2 for that open instance. The default connection behavior can be changed by using the CONNECTION= LIBNAME option.

Several SAS applications require the DB2 engine to query the DB2 system catalogs. When this type of query is required, the DB2 engine acquires a separate connection to DB2 in order to avoid contention with other applications that are accessing the DB2 system catalogs. Refer to "Accessing the DB2 System Catalogs" on page 36 for more information about accessing system catalogs.

Connections Using the Distributed Data Facility

Distributed Data Facility (DDF) is an optional product that allows OS/390 DB2 applications to access data on other 0S/390 DB2 subsystems. The DB2 engine supports DDF. To connect to a DDF remote server or location, the DB2 engine must use system-directed access. System-directed access allows one OS/390 DB2 subsystem to execute SQL statements on another OS/390 DB2 subsytem. System-directed access uses a DB2-only private protocol. The DB2 engine cannot explicitly request a connection, but instead, it performs an implicit connection when a distributed request is initiated by the SAS application. To initiate an implicit connection, the SAS option LOCATION= must be specified. When the LOCATION= option is specified, the three-level table name (location.authid.table) is used in the SQL statement that is generated by the DB2 engine. When the SQL statement that contains the three-level table name is executed, an implicit connection is made to the remote DB2 subsystem. The primary authorization ID of the initiating process must be authorized to connect to the remote location. The DB2 engine always first connects locally, then DB2 connects implicitly to a remote subsystem based that is on the location.

Connections Using the Distributed Relational Database Architecture (DRDA)

Distributed Relational Database Architecture (DRDA) is a set of protocols that enables a user to access distributed data. This enables the DB2 engine to access multiple remote tables at various locations. The tables can be distributed among multiple platforms, and both like and unlike platforms can communicate with one another. In a DRDA environment, DB2 acts as the client and/or the server. The SAS application must use the client DB2 to communicate to the server.

To connect to a DRDA remote server or location, the DB2 engine uses an explicit connection. To establish an explicit connection, the DB2 engine first connects to the local DB2 subsystem via the attachment facility (CAF or RRSAF). Then the DB2 engine issues an SQL CONNECT statement to connect from the local DB2 subsystem to the remote DRDA server prior to accessing data. The CONNECT statement is passed to the remote location after the connection is made. To initiate a connection to a DRDA remote server, you must specify the SERVER= LIBNAME option. More than one connection to a remote location is allowed, although only one connection can be active at any one time. To connect to more than one remote DRDA location, the SAS application must use one LIBNAME statement with the SERVER= option for each remote location.

Recoverable Resource Management Services Attachment Facility (RRSAF)

By default, the SAS/ACCESS engine for DB2 uses the Call Attachment Facility (CAF) to make its connections to DB2.* By setting the SAS system option DB2RRS, the DB2 engine instead uses the Recoverable Resource Manager Services Attachment Facility (RRSAF). Only one attachment facility can be used at a time, so the DB2RRS or NODB2RRS system option can only be specified when a SAS session is invoked. RRSAF is a new feature in DB2 Version 5, Release 1, and the support for it by the DB2 engine is new for Version 8 of SAS software.

The RRSAF is intended to be used by SAS servers, such as the ones used by SAS/SHARE software. RRSAF supports the ability to associate an OS/390 authorization identifier with each connection at sign on. This authorization identifier is not the same as the authorization ID that is specified in the AUTHID= data set option or SAS/ACCESS LIBNAME option. DB2 uses the RRSAF-supported authorization identifier to validate a given connection's authorization to use both DB2 and system resources, when those connections are made using the System Authorization Facility and other security products like RACF. Basically, this authorization identifier will be the userid with which you are are logged onto OS/390.

Beginning in Version 7, SAS supports multiple CAF connections for a SAS session. Thus, for a SAS server, each client can have their own connections to DB2; that is, multiple clients no longer have to share one connection. Because CAF does not support signon, however, each connection that the SAS server makes to DB2 has the OS/390 authorization identifier of the server, and not the authorization identifier of the client for which the connection is made.

With RRSAF, the SAS server makes the connections for each client and the connections have the client's OS/390 authorization identifier associated them. This is only true for clients that were authenticated by the SAS server, which occurred when the client specified a userid and password. Servers authenticate their clients when the clients provide their userids and passwords. Generally, this is the default way that servers are run. If a client connects to a SAS server without providing his userid and password, then the identifier associated with his connections will be that of the server—just like when using CAF—and not the identifier of the client.

Other than specifying DB2RRS at SAS startup, there is nothing else that needs to be done. The DB2 engine automatically signs on each connection that it makes to DB2 with either the identifier of the authenticated client or the identifier of the SAS server for non-authenticated clients. The authenticated clients have the same authorities to DB2 as they have when they run their own SAS session from their own ID and access DB2.

Accessing the DB2 System Catalogs

For various types of SAS procedures, the DB2 engine must access the DB2 system catalogs for information. This information is limited to a list of all the tables for a specific authorization identifier. There are several factors that determine what SQL query is generated to get information from the system catalogs.

- If the LIBNAME statement that references DB2 includes the AUTHID= option, or if AUTHID= was entered as a data set option, the following SQL code is generated:

```
SELECT NAME FROM SYSIBM.SYSTABLES
  WHERE (CREATOR = 'authid');
```

- If no AUTHID= option was entered, the following SQL code is generated:

* Henceforth, the SAS/ACCESS engine for DB2 will be referred to as the "DB2 engine" for brevity.

```
SELECT NAME FROM SYSIBM.SYSTABLES
  WHERE (CREATOR = "OS/390--userid");
```

The SAS procedures or applications that request the list of DB2 tables includes, but is not limited to, PROC DATASETS and PROC CONTENTS, or any application that needs a member list. If the SAS user does not have the necessary authorization to read the DB2 system catalogs, the procedure or application will fail.

Because querying the DB2 system catalogs can cause some locking contentions, the DB2 engine will initiate a separate connection for the query to the DB2 system catalogs. Once the query has completed a COMMIT WORK is executed.

For additional information about

- DB2–specific LIBNAME options, refer to “SAS/ACCESS LIBNAME Options” on page 4.
- DB2–specific data set options, refer to “SAS/ACCESS Data Set Options: DB2 Specifics” on page 8
- SAS/ACCESS LIBNAME options, refer to Chapter 3, "SAS/ACCESS LIBNAME Statement" .
- SAS/ACCESS data set options, refer to Chapter 4, "SAS/ACCESS Data Set Options" .

The correct bibliographic citation for this manual is as follows: SAS Institute Inc., *SAS/ACCESS® Software for Relational Databases: Reference, Version 8 (DB2® under OS/390® Chapter)*, Cary, NC: SAS Institute Inc., 1999.

SAS/ACCESS® Software for Relational Databases: Reference, Version 8 (DB2® under OS/390® Chapter)

ISBN 1-58025-539-6

SAS Institute Inc., SAS Campus Drive, Cary, North Carolina 27513.

1st printing, October 1999