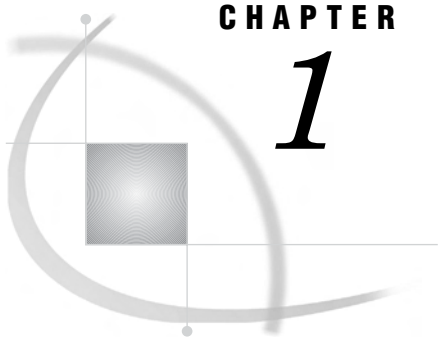

CHAPTER

1

SAS/ACCESS for DB2 under UNIX and PC Hosts

Introduction to the SAS/ACCESS Interface to DB2 under UNIX and PC Hosts

This document includes details *only* about the SAS/ACCESS interface to DB2 under UNIX and PC Hosts. It should be used as a supplement to the generic SAS/ACCESS documentation, *SAS/ACCESS for Relational Databases: Reference*.

LIBNAME Statement Specifics for DB2 under UNIX and PC Hosts

This section describes the LIBNAME statement as supported by the interface to DB2 under UNIX and PC hosts. For a complete description of this feature, see the LIBNAME statement section in *SAS/ACCESS for Relational Databases: Reference*. The DB2 under UNIX and PC hosts specific syntax for the LIBNAME statement is as follows:

LIBNAME *libref* **db2** <*connection-options*> <*LIBNAME-options*>;

Arguments

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

db2
is the SAS/ACCESS engine name for the interface to DB2 under UNIX and PC hosts.

connection-options
provide connection information and control how SAS manages the timing and concurrence of the connection to the DBMS. There are several ways to connect to DB2 when you are using the LIBNAME statement. Use *only one* of the following methods for each connection since they are mutually exclusive:

- specify USER=, PASSWORD=, and DATASRC=
- specify COMPLETE=
- specify NOPROMPT=
- specify PROMPT=
- specify READBUFF=
- specify REQUIRED=.

Definitions of these connection options are provided below.

USER=<'>*user-name*<'>
enables you to connect to a DB2 database with a user ID that is different from the default ID.

USER= is optional. If you specify USER=, you must also specify PASSWORD=. If USER= is omitted, your default user ID for your operating environment is used.

PASSWORD=<'>*password*<'>
specifies the DB2 password that is associated with your DB2 user ID.

PASSWORD= is optional. If you specify USER=, you must specify PASSWORD=.

DATASRC=<'>*data-source-name*<'>

specifies the DB2 data source or database to which you want to connect.

DATASRC= is optional. If you omit it, you connect by using a default environment variable.

DSN= and DATABASE= are aliases for this option.

COMPLETE=<'>*CLI-connection-string*<'>
specifies connection information for your data source or database for PCs only. Separate multiple options with a semicolon. When a successful connection is made, the complete connection string is returned in the SYSDBMSG macro variable.

If you do not specify enough correct connection options, you are prompted with a dialog box that displays the values from the COMPLETE= connection string. You can edit any field before you connect to the data source.

This option is not available on UNIX platforms. See your DB2 documentation for more details.

NOPROMPT=<'>*CLI-connection-string*<'>
specifies connection information for your data source or database. Separate multiple options with a semicolon.

If you do not specify enough correct connection options, an error is returned (no dialog box is displayed).

PROMPT=<'> *CLI-connection-string*<'>
specifies connection information for your data source or database for PCs only. Separate multiple options with a semicolon. When a successful connection is made, the complete connection string is returned in the SYSDBMSG macro variable.

PROMPT= does not immediately attempt to connect to the DBMS. Instead, it displays a dialog box that contains the values that you entered in the PROMPT= connection string. You can edit values or enter additional values in any field before you connect to the data source.

This option is not available on UNIX platforms.

READBUFF= *number-of-rows*
READBUFF is used to control the number of rows that are fetched during each read operation. By default, an optimal value is calculated by the SAS/ACCESS to DB2 engine based on the row length of your data. You should not have to specify this option unless you suspect problems related to fetching multiple rows from the DB2 database.

REQUIRED=<'>*CLI-connection-string*<'>
specifies connection information for your data source or database for PCs only. Separate multiple options with a semicolon. When a successful connection is made, the complete connection string is returned in the SYSDBMSG macro variable.

If you do not specify enough correct connection options, a dialog box prompts you for the connection options. REQUIRED= only allows you to modify required fields in the dialog box.

This option is not available on UNIX platforms.

LIBNAME-options
define how DBMS objects are processed by SAS. Some LIBNAME options can enhance performance; others determine locking or naming behavior. The following table describes which LIBNAME options are supported for DB2 under UNIX and PC hosts, and presents default values where applicable. See the section about the LIBNAME statement in *SAS/ACCESS for Relational Databases: Reference* for detailed information about these options.

Table 1.1 SAS/ACCESS LIBNAME Options for DB2 under UNIX and PC Hosts

Option	Default Value
ACCESS=	none
AUTOCOMMIT=	varies with transaction type
CONNECTION=	SHAREDREAD
CONNECTION_GROUP=	none
CURSOR_TYPE=	DYNAMIC
DBCOMMIT=	1000 (insert); 0 (update); 10000 (bulk load)
DBCONINIT=	none
DBCONTERM=	none
DBCREATE_TABLE_OPTS=	none
DBGEN_NAME=	DBMS
DBINDEX=	YES
DBLIBINIT=	none
DBLIBTERM=	none
DBMAX_TEXT=	1024
DBNULLKEYS=	YES
DBPROMPT=	NO
DBSLICEPARM=	THREADED_APPS,2 or 3
DEFER=	NO
DIRECT_EXE=	none
DIRECT_SQL=	YES
IGNORE_ READ_ONLY_COLUMNS=	NO
IN=	none
INSERTBUFF=	automatically calculated based upon row length
MULTI_DATASRC_OPT=	NONE
PRESERVE_COL_NAMES=	NO (see “DB2 under UNIX and PC Hosts Naming Conventions” on page 19)
PRESERVE_TAB_NAMES=	NO (see “DB2 under UNIX and PC Hosts Naming Conventions” on page 19)
QUERY_TIMEOUT=	0
READBUFF=	automatically calculated based upon row length
READ_ISOLATION_LEVEL=	set by the user in the DB2Cli.ini file (see “Locking for DB2 under UNIX and PC Hosts Interface” on page 15)
READ_LOCK_TYPE=	ROW
REREAD_EXPOSURE=	NO
SCHEMA=	your user ID
SPOOL=	YES

Option	Default Value
SQL_FUNCTIONS=	NONE
STRINGDATES=	NO
UPDATE_ISOLATION_LEVEL=	CS (see "Locking for DB2 under UNIX and PC Hosts Interface" on page 15)
UPDATE_LOCK_TYPE=	ROW
UTILCONN_TRANSIENT=	YES

DB2 UNIX/PC LIBNAME Statement Example

In the following example, the libref MyDBLib uses the DB2 engine and the NOPROMPT= option to connect to a DB2 database. PROC PRINT is used to display the contents of the DB2 table Customers.

```
libname mydblib db2
   noprompt="dsn=userdsn;uid=testuser;pwd=testpass;";

proc print data=mydblib.customers;
   where state='CA';
run;
```

Data Set Options for DB2 under UNIX and PC Hosts

The following table describes the data set options that are supported for DB2 under UNIX and PC hosts, and provides default values where applicable. See the section about data set options in *SAS/ACCESS for Relational Databases: Reference* for detailed information about these options.

Table 1.2 SAS/ACCESS Data Set Options for DB2 under UNIX and PC Hosts

Option	Default Value
BL_CODEPAGE=	the window's codepage ID
BL_COPYLOCATION=	none
BL_DATAFILE=	the current directory
BL_DELETE_DATAFILE=	YES
BL_INDEXING_MODE=	AUTOSELECT
BL_LOAD_REPLACE=	NO
BL_LOG=	the current directory
BL_METHOD=	none
BL_OPTIONS=	none
BL_RECOVERABLE=	NO
BL_REMOTEFILE=	none
BL_SERVER_DATAFILE=	same as BL_DATAFILE
BL_WARNING_COUNT=	2147483646

Option	Default Value
BULKLOAD=	NO
CURSOR TYPE=	LIBNAME option setting
DBCOMMIT=	LIBNAME option setting
DBCONDITION=	none
DBCREATE_TABLE_OPTS=	LIBNAME option setting
DBFORCE=	NO
DBGEN_NAME=	DBMS
DBINDEX=	LIBNAME option setting
DBKEY=	none
DBLABEL=	NO
DBMASTER=	none
DBMAX_TEXT=	1024
DBNULL=	_ALL_=YES
DBNULLKEYS=	LIBNAME option setting
DBPROMPT=	LIBNAME option setting
DBSASTYPE=	see “Data Types for DB2 under UNIX and PC Hosts” on page 20
DBSLICE=	none
DBSLICEPARM=	THREADED_APPS,2 or 3
DBTYPE=	see “Data Types for DB2 under UNIX and PC Hosts” on page 20
ERRLIMIT=	1
IGNORE_ READ_ONLY_COLUMNS=	NO
IN=	LIBNAME option setting
INSERTBUFF=	LIBNAME option setting
NULLCHAR=	SAS
NULLCHARVAL=	a blank character
PRESERVE_COL_NAMES=	LIBNAME option setting
QUERY_TIMEOUT=	LIBNAME option setting
READ_ISOLATION_LEVEL=	LIBNAME option setting
READ_LOCK_TYPE=	LIBNAME option setting
READBUFF=	LIBNAME option setting
SASDATEFMT=	none
SCHEMA=	LIBNAME option setting

Option	Default Value
UPDATE_ISOLATION_LEVEL=	LIBNAME option setting
UPDATE_LOCK_TYPE=	LIBNAME option setting

Pass-Through Facility Specifics for DB2 under UNIX and PC Hosts

See the section about the Pass-Through Facility in *SAS/ACCESS for Relational Databases: Reference* for general information about this feature.

The Pass-Through Facility specifics for DB2 under UNIX and PC Hosts are as follows:

- The *dbms-name* is **DB2**.
- The CONNECT statement is required.
- You can connect to only one DB2 database at a time. However, you can use multiple CONNECT statements to connect to multiple DB2 data sources by using the *alias* argument to distinguish your connections.
- The *database-connection-arguments* for the CONNECT statement are identical to its LIBNAME connection options.
- The following LIBNAME options are available with the CONNECT statement:

  AUTOCOMMIT=
  CURSOR_TYPE=
  QUERY_TIMEOUT=
  READBUFF=
  READ_ISOLATION_LEVEL=

  See the section about the LIBNAME statement in *SAS/ACCESS for Relational Databases: Reference* for information about these options.

Examples

The following example connects to the SAMPLE database and sends it two EXECUTE statements to process.

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

The following example connects to the SAMPLE database by using an alias (DB1) and performs a query, shown in italic type, on the SASDEMO.CUSTOMERS table.

```
proc sql;
   connect to db2 as db1 (database=sample);
   select *
      from connection to db1
         (select * from sasdemo.customers
             where customer like '1%');
   disconnect from db1;
quit;
```

Autopartitioning Scheme for DB2 under UNIX and PC Hosts

See the section about threaded reads in *SAS/ACCESS for Relational Databases: Reference* for general information about this feature.

Overview

The autopartitioning method available for the SAS/ACCESS interface to DB2 for UNIX and PC hosts is a MOD function method as described in the section about autopartitioning techniques in *SAS/ACCESS for Relational Databases: Reference*.

Autopartitioning Restrictions

The interface to DB2 under UNIX and PC hosts restricts which columns can be used for the partitioning column during the autopartitioning phase. Columns are partitioned as follows:

- INTEGER and SMALLINT columns are given preference.
- The other DB2 numeric columns may be used for partitioning, provided that the precision minus the scale of the column is between 0 and 10. That is, 0<(precision-scale)<10.

Nullable Columns

If a nullable column is selected for partitioning, then the SQL statement **OR<column-name>IS NULL** is appended to the end of the SQL code that is generated for the threaded reads. This ensures that any possible NULL values are returned in the result set.

Using WHERE Clauses

Autopartitioning does not select a column to be the partitioning column if it appears in the WHERE clause. For instance, the following data step would not be able to use a threaded read to retrieve the data since all of the numeric columns in the table (see the table definition as described in "Using DBSLICE=" on page 9) are in the WHERE clause:

```
data work.locemp;
set trlib.MYEMPS;
where EMPNUM<=30 and ISTENURE=0 and
 SALARY<=35000 and NUMCLASS>2;
run;
```

Using DBSLICEPARM=

When using autopartitioning, and DBSLICEPARM= does not specify a maximum number of threads to use for the threaded read, SAS/ACCESS to DB2 under UNIX and PC hosts defaults to three threads.

Using DBSLICE=

You can achieve the best possible performance when using threaded reads by specifying a DB2-specific DBSLICE= data set option in your SAS operation. This statement is especially true if your DB2 data is evenly distributed across multiple partitions in a DB2 Enterprise Extended Edition (EEE) database system. When creating a DB2 table under the DB2 EEE model, you can specify the partitioning key you want to use by appending the clause **PARTITIONING KEY(*column-name*)** to your CREATE TABLE statement. Inside the SAS environment, this can be accomplished with the LIBNAME option DBCREATE_TABLE_OPTS=, as follows:

```
/*points to a triple node server*/
libname trlib2 db2 user=db2user pw=db2pwd db=sample3c
DBCREATE_TABLE_OPTS='PARTITIONING KEY(EMPNUM);

proc delete data=trlib.MYEMPS1;
run;

data trlib.myemps(drop=morf whatstate
   DBTYPE=(HIREDATE="date" SALARY="numeric(8,2)"
   NUMCLASS="smallint" GENDER="char(1)" ISTENURE="numeric(1)" STATE="char(2)"
   EMPNUM="int NOT NULL Primary Key"));
format HIREDATE mmddyy10.;
do EMPNUM=1 to 100;
     morf=mod(EMPNUM,2)+1;
     if(morf eq 1) then
         GENDER='F';
     else
         GENDER='M';
     SALARY=(ranuni(0)*5000);
     HIREDATE=int(ranuni(13131)*3650);
     whatstate=int(EMPNUM/5);
     if(whatstate eq 1) then
         STATE='FL';
     if(whatstate eq 2) then
         STATE='GA';
     if(whatstate eq 3) then
         STATE='SC';
     if(whatstate eq 4) then
         STATE='VA';
     else
         state='NC';
     ISTENURE=mod(EMPNUM,2);
     NUMCLASS=int(EMPNUM/5)+2;
     output;
end;
run;
```

After the table MYEMPS is created on this three node database, one third of the rows will reside on each of the three nodes.

The optimization of the threaded read against this partitioned table depends upon the location of the DB2 partitions. If the DB2 partitions reside on the same machine, you can use DBSLICE= in conjunction with the DB2 NODENUMBER function in the WHERE clause:

```
proc print data=trlib2.MYEMPS(DBSLICE=("NODENUMBER(EMPNO)=0"
   "NODENUMBER(EMPNO)=1" "NODENUMBER(EMPNO)=2"));
run;
```

If the DB2 partitions reside on different physical machines, you can usually obtain the best results by using the DBSLICE= option with the SERVER= syntax inaddition to the DB2 NODENUMBER function in the WHERE clause.

In the following example, the DBSLICE= option contains the DB2-specific partitioning information, and Sample3a, Sample3b, and Sample3c are DB2 database aliases that point to individual DB2 EEE database nodes that exist on separate physical machines. For more information about the configuration of these nodes, refer to "Configuring DB2 EEE Nodes on Physically Partitioned Databases" on page 10.

```
proc print data=trlib2.MYEMPS(DBSLICE=(sample3a="NODENUMBER(EMPNO)=0"
   samble3b="NODENUMBER(EMPNO)=1" sample3c="NODENUMBER(EMPNO)=2"));
run;
```

Note that NODENUMBER is not required in order to use threaded reads for SAS/ACCESS to DB2. The methods and examples described in DBSLICE= work well in instances where the table you want to read is not stored in multiple partitions to DB2. These methods also give you full control over which column is used to execute the threaded read. For instance, if the STATE column in your employee table only contains a few distinct values, you can tailor your DBSLICE= clause accordingly:

```
data work.locemp;
set trlib2.MYEMPS (DBSLICE=("STATE='GA'"
    "STATE='SC'" "STATE='VA'" "STATE='NC'"));
where EMPNUM<=30 and ISTENURE=0 and SALARY<=35000 and NUMCLASS>2;
run;
```

Configuring DB2 EEE Nodes on Physically Partitioned Databases

Assuming that the database SAMPLE is partitioned across three different machines, you can create a database alias for it at each node from the DB2 Command Line Processor by issuing the following commands:

```
catalog tcpip node node1 remote <hostname> server 50000
catalog tcpip node node2 remote <hostname> server 50000
catalog tcpip node node3 remote <hostname> server 50000
catalog database sample as samplea at node node1
catalog database sample as sampleb at node node2
catalog database sample as samplec at node node3
```

This enables SAS/ACCESS to DB2 to access the data for the SAMPLE table directly from each node. For more information about configuring DB2 EEE to use multiple physical partitions, see the *DB2 Administrators Guide*.

Temporary Table Support for DB2 under UNIX and PC Hosts

See the section on the temporary table support in *SAS/ACCESS for Relational Databases: Reference* for general information about this feature.

Establishing a Temporary Table

To make full use of temporary tables, the CONNECTION=GLOBAL connection option is necessary. This option allows a single connection to be used across SAS DATA steps and procedure boundaries as well as be shared between LIBNAME statements and the Pass-Through Facility. Since a temporary table only exists within a single connection, you must be able to share this single connection between all of the steps that reference the temporary table. The temporary table cannot be referenced from any other connection.

The type of temporary table that is used for this processing is created using the DECLARE GLOBAL TEMPORARY TABLE statement with the ON COMMIT PRESERVE ROWS clause. This kind of temporary table lasts for the life of the connection, unless explicitly dropped, and retains its rows of data beyond commit points.

It is important to note that DB2 places all global temporary tables in the SESSION schema. Therefore, in order to reference these temporary tables within SAS, you must explicitly provide the SESSION schema in Pass-Through SQL statements or use the SCHEMA= LIBNAME option with a value of SESSION.

Currently, the only supported way to create a temporary table is to use a PROC SQL statement. In order to use both the Pass-Through Facility and librefs to reference a temporary table, you need to specify a LIBNAME statement before the PROC SQL step. This enables the global connection to persist across SAS steps, even multiple PROC SQL steps. For example:

```
libname temp db2 database=sample user=myuser password=mypwd
        schema=SESSION connection=global;

proc sql;
   connect to db2 (db=sample user=myuser pwd=mypwd connection=global);
   execute (declare global temporary table temptab1 like other.table
            on commit PRESERVE rows not logged) by db2;
quit;
```

At this point, you can refer to the temporary table by using the libref Temp or by using the CONNECTION=GLOBAL option with a PROC SQL step.

Terminating a Temporary Table

You can drop a temporary table at any time, or allow it to be implicitly dropped when the connection is terminated. Temporary tables do not persist beyond the scope of a single connection.

Examples

The following examples assume there is a DeptInfo table on the DBMS that has all of your department information. They also assume that you have a SAS data set with join criteria that you want to use to get certain rows out of the DeptInfo table, and another SAS data set with updates to the DeptInfo table.

The following librefs and temporary tables are used:

```
libname saslib base 'SAS-Data-Library';
libname dept db2 db=sample user=myuser pwd=mypwd connection=global;
libname temp db2 db=sample user=myuser pwd=mypwd connection=global
        schema=SESSION;
/* Note that the temporary table has a schema of SESSION */

proc sql;
   connect to db2 (db=sample user=myuser pwd=mypwd connection=global);
   execute (declare global temporary table
            temptab1 (dname char(20), deptno int)
            on commit PRESERVE rows not logged) by db2;
quit;
```

The following example demonstrates how to take a heterogeneous join and use a temporary table to perform a homogeneous join on the DBMS (as opposed to reading the DBMS table into SAS to perform the join). Using the table created above, the SAS data is copied into the temporary table to perform the join.

```
proc sql;
   connect to db2 (db=sample user=myuser pwd=mypwd connection=global);
   insert into temp.temptab1 select * from saslib.joindata;
   select * from dept.deptinfo info, temp.temptab1 tab
        where info.deptno = tab.deptno;
   /* remove the rows for the next example */
   execute (delete from session.temptab1) by db2;
   quit;
```

In the following example, transaction processing on the DBMS occurs using a temporary table as opposed to using either DBKEY= or MULTI_DATASRC_OPT=IN_CLAUSE with a SAS data set as the transaction table.

```
connect to db2 (db=sample user=myuser pwd=mypwd connection=global);
   insert into temp.temptab1 select * from saslib.transdat;
   execute (update deptinfo d set deptno = (select deptno from session.temptab1)
            where d.dname = (select dname from session.temptab1)) by db2;
quit;
```

DBLOAD Procedure Specifics for DB2 under UNIX and PC Hosts

See the section about the DBLOAD procedure in *SAS/ACCESS for Relational Databases: Reference* for general information about this feature.

The DB2 under UNIX and PC Hosts interface supports all of the DBLOAD procedure statements in batch mode. The DBLOAD procedure specifics for DB2 under UNIX and PC hosts are as follows:

- DBMS= value is **DB2**.
- PROC DBLOAD uses the following database description statements:

  IN= <'>*database-name*<'>;
  specifies the name of the database in which you want to store the new DB2 table. The IN= statement is required and must immediately follow the PROC DBLOAD statement. The *database-name* is limited to eight characters. DATABASE= is an alias for the IN= statement.

  The database that you specify must already exist. If the database name contains the following special characters (_,$,@,#), you must enclose it in

quotation marks. However, DB2 recommends against using special characters in database names.

USER= <'>*username*<'>;
enables you to connect to a DB2 database, such as Microsoft SQL Server or AS/400, with a user ID that is different from the default login ID.

USER= is optional in the interface to DB2 UNIX/PC. If you specify USER=, you must also specify PASSWORD=. If USER= is omitted, your default user ID is used.

PASSWORD= <'>*password*<'>;
specifies the password that is associated with your user ID.

PASSWORD= is optional in the interface to DB2 under UNIX and PC hosts because users have default user IDs. If you specify USER=, however, you must specify PASSWORD=.

Note: If you do not wish to enter your DB2 password in uncoded text on this statement, see PROC PWENCODE in *Base SAS Procedures Guide* for a method to encode it. △

- The TABLE= statement is as follows:

  TABLE= <'><*schema-name.*>*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. If you use quotation marks, the name is case-sensitive. The TABLE= statement is required.

  The *schema-name* is a person's name or group ID that is associated with the DB2 table. The schema name is limited to eight characters.

- The NULLS statement is as follows:

  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.

  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.

Examples

The following example creates a new DB2 table, SASDEMO.EXCHANGE, from the MYDBLIB.RATEOFEX data file. You must be granted the appropriate privileges in order to create new DB2 tables or views.

```
proc dbload dbms=db2 data=mydblib.rateofex;
   in='sample';
   user='testuser';
   password='testpass';
   table=sasdemo.exchange;
      rename fgnindol=fgnindollars
         4=dollarsinfgn;
   nulls updated=n fgnindollars=n
         dollarsinfgn=n country=n;
   load;
run;
```

The following example sends only a DB2 SQL GRANT statement to the SAMPLE database and does not create a new table. Therefore, the TABLE= and LOAD statements are omitted.

```
proc dbload dbms=db2;
   in='sample';
   sql grant select on sasdemo.exchange
      to testuser;
run;
```

Passing SAS Functions to DB2 under UNIX and PC Hosts

The interface to DB2 under UNIX and PC hosts passes the following SAS functions to DB2 for processing (if the DBMS driver/client that you are using supports the function). Where the DB2 function name is different than the SAS function name, the DB2 name appears in parentheses. See the section about optimizing SQL usage in *SAS/ACCESS for Relational Databases: Reference* for information.

ABS
ARCOS (ACOS)
ARSIN (ASIN)
ATAN
AVG
CEIL (CEILING)
COS
COSH
COUNT (COUNT_BIG)
EXP
FLOOR
LOG
LOG10
LOWCASE
MAX
MIN
MOD
SIGN
SIN
SINH
SQRT
SUM
TAN
TANH
LOWCASE (LCASE)
UPCASE (UCASE)

If SQL_FUNCTIONS=ALL, the following options are passed down:
BYTE (CHAR)

COMPRESS (REPLACE)
DATE (CURDATE)
DATETIME (NOW)
DAY (DAYOFMONTH)
HOUR
INDEX (LOCATE)
LENGTH
MINUTE
MONTH
QTR (QUARTER)
REPEAT
SECOND
SOUNDEX
SUBSTR (SUBSTRING)
TIME (CURTIME)
TODAY (CURDATE)
TRIMN (RTRIM)
TRANWRD (REPLACE)
WEEKDAY (DAYOFWEEK)
YEAR

Passing Joins to DB2 under UNIX and PC Hosts

In order for a multiple libref join to pass to DB2 under UNIX and PC hosts, all of the following components of the LIBNAME statements must match exactly:

user ID

password

Update_Isolation_Level
(if specified)

Read_Isolation_Level
(if specified)

qualifier

datasource

PROMPT
must *not* be
specified

See the section about performance considerations in *SAS/ACCESS for Relational Databases: Reference* for more information about when and how SAS/ACCESS passes joins to the DBMS.

Locking for DB2 under UNIX and PC Hosts Interface

The following LIBNAME and data set options enable you to control how the interface to DB2 under UNIX and PC hosts handles locking. See the section about the LIBNAME

statement in *SAS/ACCESS for Relational Databases: Reference* for additional information about these options.

READ_LOCK_TYPE= ROW | TABLE

UPDATE_LOCK_TYPE= ROW | TABLE

READ_ISOLATION_LEVEL= RR | RS | CS | UR
The DB2 database manager supports the RR, RS, CS, and UR isolation levels defined in the following table. Regardless of the isolation level, the database manager places exclusive locks on every row that is inserted, updated, or deleted. Thus, all isolation levels ensure that any row that is changed by this application process during a unit of work is not changed by any other application process until the unit of work is complete.

Table 1.3 Isolation Levels for DB2 under UNIX and PC Hosts

Isolation Level	Definition
RR (Repeatable Read)	no dirty reads, no nonrepeatable reads, no phantom reads
RS (Read Stability)	no dirty reads, no nonrepeatable reads; does allow phantom reads
CS (Cursor Stability)	no dirty reads; does allow nonrepeatable reads and phantom reads
UR (Uncommitted Read)	allows dirty reads, nonrepeatable reads, and phantom reads

The terms in the table are defined as follows:

- *Dirty reads* — A transaction that exhibits this phenomenon has very minimal isolation from concurrent transactions. In fact, it is able to see changes made that are by those concurrent transactions even before they commit.

  For example, suppose that transaction T1 performs an update on a row, transaction T2 then retrieves that row, and transaction T1 then terminates with rollback. Transaction T2 has then seen a row that no longer exists.

- *Nonrepeatable reads* — If a transaction exhibits this phenomenon, it is possible that it might read a row once and, if it attempts to read that row again later in the course of the same transaction, the row might have been changed or even deleted by another concurrent transaction. Therefore, the read is not (necessarily) repeatable.

  For example, suppose that transaction T1 retrieves a row, transaction T2 then updates that row, and transaction T1 then retrieves the same row again. Transaction T1 has now retrieved the same row twice but has seen two different values for it.

- *Phantom reads* — When a transaction exhibits this phenomenon, a set of rows that it reads once might be a different set of rows if the transaction attempts to read them again.

  For example, suppose that transaction T1 retrieves the set of all rows that satisfy some condition. Suppose that transaction T2 then inserts a new row that satisfies that same condition. If transaction T1 now repeats its retrieval request, it sees a row that did not previously exist, a “phantom.”

UPDATE_ISOLATION_LEVEL = CS | RS | RR
The DB2 database manager supports the CS, RS, and RR isolation levels defined in the preceding table. Uncommitted reads are not allowed with this option.

DB2 under UNIX and PC Hosts Bulk Loading

Bulk loading is the fastest way to insert large numbers of rows into a DB2 table. Using this facility enables you to insert rows two to ten times more quickly than using regular SQL insert statements. You must specify BULKLOAD=YES in order to use the bulk load facility.

The interface to DB2 under UNIX and PC hosts has three bulk loading methods available; IMPORT, LOAD, and CLI LOAD. The method used is determined by the data set options BL_REMOTE_FILE= and BL_METHOD=. The following are brief descriptions of these three methods.

- In order to use the LOAD method, you must have system administrator authority, database administrator authority, or load authority on the database and the insert privilege on the table being loaded.

  This method also requires that the client and server machines are able to read and write files to a common location, such as a mapped network drive or an NFS directory. To use this method, specify the BL_REMOTE_FILE= option.

  Note:

  - To avoid possible tablespace issues, the DB2 client and database should be DB2 Version 6 or later.
  - Since SAS/ACCESS to DB2 uses the PC/IXF file format to transfer data to the DB2 LOAD utility, this method cannot be used to load data into partitioned databases.

  △

  The bulk loading options available with the LOAD method are listed below. See the section about data set options in *SAS/ACCESS for Relational Databases: Reference* for detailed information about these options.

  BL_CODEPAGE=

  BL_DATAFILE=

  BL_DELETE_DATAFILE=

  BL_LOG=

  The log file contains a summary of load information and error descriptions. On most platforms, the default file name takes the form *BL_<table>_<unique-ID>*.log where

  table is the table name

  unique-ID is a number used to prevent collisions in the event of two or more simultaneous bulk loads of a particular table. The SAS/ACCESS engine generates the number.

  BL_OPTIONS=

  BL_REMOTE_FILE=

  BL_SERVER_DATAFILE =

  BL_WARNING_COUNT=.

- The IMPORT method does not offer the same level of performance as the LOAD method, but it is available to all users who have insert privileges on the tables being loaded. The IMPORT method does not require that the server and client have a common location in order to access the data file. If you do not specify BL_REMOTE_FILE=, the IMPORT method is automatically used.

  The bulk loading options available with the IMPORT method are listed below. See the section about data set options in *SAS/ACCESS for Relational Databases: Reference* for detailed information about these options.

BL_CODEPAGE=

BL_DATAFILE=

BL_DELETE_DATAFILE=

BL_LOG=

BL_OPTIONS=.

- The CLI LOAD method is an interface to the standard DB2 LOAD utility, which gives the added performance of using LOAD but without setting additional options for the bulk load. This method requires the same privileges as the LOAD method, and is only available in DB2 Version 7 FixPak 4 and later clients and servers. If your client and server can support the CLI LOAD method, you will generally see the best performance by using it. The CLI LOAD method can also be used to load data into a partitioned DB2 database for client and database nodes that are DB2 Version 8.1 or later. To use this method, specify BL_METHOD=CLILOAD as a data set option. The bulk loading options available with the CLI LOAD method are listed below:

  BL_COPY_LOCATION=

  BL_INDEXING_MODE=

  BL_LOAD_REPLACE=

  BL_LOG=

  BL_METHOD=

  BL_OPTIONS=

  BL_RECOVERABLE=

  BL_REMOTE_FILE=

For more information about the differences between IMPORT, LOAD, and CLI LOAD, refer to the *DB2 Data Movement Utilities Guide and Reference*.

Maximizing Load Performance for DB2 under UNIX and PC Hosts

The following tips will help you optimize LOAD performance when you are using the DB2 bulk load facility:

- Specifying BL_REMOTE_FILE= causes the loader to use the DB2 LOAD utility, which is much faster than the IMPORT utility, but it requires database administrator authority.
- Performance might suffer if your setting for DBCOMMIT= is too low. Increase the default (which is 10000 when BULKLOAD=YES) for much better performance.
- Increasing the DB2 tuning parameters, such as Utility Heap and I/O characteristics, improves performance. These parameters are controlled by your database or server administrator.
- When using the IMPORT utility, specify BL_OPTIONS="COMPOUND=x" where x is a number between 1 and 7 on Windows, and between 1 and 100 on UNIX. This causes the IMPORT utility to insert multiple rows for each execute instead of 1 row per execute.
- When using the LOAD utility on a multi-processor or multi-node DB2 server, specify BL_OPTIONS="ANYORDER" to improve performance. Note that this might cause the entries in the DB2 log to be out of order (because it enables DB2 to insert the rows in a order that is different from how they appear in the loader data file).

Examples

The following example shows how to use a SAS data set, SASFLT.FLT98, to create and load a large DB2 table, FLIGHTS98. Because the code specifies BULKLOAD=YES and BL_REMOTE_FILE= is omitted, this load uses the DB2 IMPORT command.

```
libname sasflt 'SAS-data-library';
libname db2_air db2 user=louis using=fromage
        database='db2_flt' schema=statsdiv;

proc sql;
create table db2_air.flights98
       (bulkload=YES bl_options='compound=7 norowwarnings')
        as select * from sasflt.flt98;
quit;
```

The BL_OPTIONS= option passes DB2 file type modifiers to DB2. The **norowwarnings** modifier indicates that all row warnings about rejected rows are to be suppressed.

The following example shows how to append the SAS data set, SASFLT.FLT98 to a preexisting DB2 table, ALLFLIGHTS. Because the code specifies BULKLOAD=YES and BL_REMOTE_FILE=, this load uses the DB2 LOAD command.

```
proc append base=db2_air.allflights
 (BULKLOAD=YES
  BL_REMOTE_FILE='/tmp/tmpflt'
  BL_LOG='/tmp/fltdata.log'
  BL_DATAFILE='/nfs/server/tmp/fltdata.ixf'
  BL_SERVER_DATAFILE='/tmp/fltdata.ixf')
data=sasflt.flt98;
run;
```

Here, BL_REMOTE_FILE= and BL_SERVER_DATAFILE= are paths relative to the server. BL_LOG= and BL_DATAFILE= are paths relative to the client.

The following example shows how to use the SAS data set SASFLT.ALLFLIGHTS to create and load a large DB2 table, ALLFLIGHTS. Because the code specifies BULKLOAD=YES and BL_METHOD=CLILOAD, this operation uses the DB2 CLI LOAD interface to the LOAD command.

```
data db2_air.allflights(BULKLOAD=YES BL_METHOD=CLILOAD);
set sasflt.allflights;
run;
```

DB2 under UNIX and PC Hosts Naming Conventions

The PRESERVE_TAB_NAMES= and PRESERVE_COL_NAMES= options determine how the interface to DB2 under UNIX and PC hosts handles case-sensitivity. DB2 is case-insensitive and all names default to uppercase. See the section about the LIBNAME statement in *SAS/ACCESS for Relational Databases: Reference* for additional information about these options.

DB2 objects include tables, views, columns, and indexes. Use the following naming conventions for them:

- A name can start with a letter or one of the following symbols: the dollar sign ($), the number (or pound) sign (#), or the at symbol (@).

- A name can be from 1 to 18 characters long.
- A name can contain the letters A through Z, any valid letter with an accent (such as a), the digits 0 through 9, the underscore (_), the dollar sign ($), the number or pound sign (#), or the at symbol (@).
- A name is not case-sensitive (for example, the table name CUSTOMERS is the same as Customers), but object names are converted to uppercase when typed. If a name is enclosed in quotation marks, then the name is case-sensitive.
- A name cannot be a DB2 or an SQL reserved word, such as WHERE or VIEW.
- A name cannot be the same as another DB2 object that has the same type.

Schema and database names have similar conventions, except that they are each limited to eight characters. For more information, see your DB2 SQL reference manual.

Data Types for DB2 under UNIX and PC Hosts

Every column in a table has a name and a data type. The data type tells DB2 how much physical storage to set aside for the column and the form in which the data is stored. DB2 uses IBM SQL data types. For more information about DB2 data types, and to determine which data types are available for your version of DB2, see your DB2 SQL reference manual.

Note: SAS/ACCESS does not support the BLOB, CLOB, and DBCLOB DB2 data types. △

String Data

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

VARCHAR(*n*)
specifies a varying-length column for character string data. The maximum length of the string is 4000 characters. If the length is greater than 254, the column is a long-string column. SQL imposes some restrictions on referencing long-string columns. For more information about these restrictions, see your IBM documentation.

LONG VARCHAR
specifies a varying-length column for character string data. The maximum length of a column of this type is 32700 characters. A LONG VARCHAR column cannot be used in certain functions, subselects, search conditions, and so forth. For more information about these restrictions, see your IBM documentation.

GRAPHIC(*n*)
specifies a fixed-length column for graphic string data. *n* specifies the number of double-byte characters and can range from 1 to 127. If *n* is not specified, the default length is 1.

VARGRAPHIC(*n*)
specifies a varying-length column for graphic string data. *n* specifies the number of double-byte characters and can range from 1 to 2000.

LONG VARGRAPHIC

specifies a varying-length column for graphic-string data. *n* specifies the number of double-byte characters and can range from 1 to 16350.

Numeric Data

BIGINT
specifies a big integer. Values in a column of this type can range from -9223372036854775808 to +9223372036854775807.

SMALLINT
specifies a small integer. Values in a column of this type can range from -32768 through +32767.

INTEGER
specifies a large integer. Values in a column of this type can range from -2147483648 through +2147483647.

FLOAT | DOUBLE | DOUBLE PRECISION
specifies a floating-point number that is 64 bits long. Values in a column of this type can range from –1.79769E+308 to –2.225E–307 or +2.225E–307 to +1.79769E+308, or they can be 0. (This data type is stored the same way that SAS stores its numeric data type. Therefore, numeric columns of this type require the least processing when they are being accessed by SAS.)

DECIMAL | DEC | NUMERIC | NUM
specifies a mainframe packed decimal number with an implicit decimal point. The position of the decimal point is determined by the precision and scale of the number. The scale, which is the numbers to the right of the decimal point, cannot be negative or greater than the precision. The maximum precision is 31 digits. Note that numbers that require decimal precision greater than 15 digits might be subject to rounding and conversion errors.

Dates, Times, and Timestamps

SQL date and time data types are collectively called datetime values. The SQL data types for dates, times, and timestamps are listed here. Be aware that columns of these data types can contain data values that are out of range for SAS.

DATE
specifies date values in various formats, as determined by the country code of the database. For example, the default format for the United States is *mm-dd-yyyy* and the European standard format is *dd.mm.yyyy*. The range is 01-01-0001 to 12-31-9999. A date always begins with a digit, is at least eight characters long, and is represented as a character string. For example, in the U.S. default format, January 25, 1991, would be formatted as 01-25-1991.

The entry format can vary according to the edit codes that are associated with the field. For more information about edit codes, see your IBM documentation.

TIME
specifies time values in a three part format. The values range from 0 to 24 for hours (*hh*) and from 0 to 59 for minutes (*mm*) and seconds (*ss*). The default form for the United States is *hh:mm:ss*, and the IBM European standard format for time is *hh.mm[.ss]*. For example, in the U.S. default format 2:25 p.m. would be formatted as 14:25:00.

The entry format can vary according to the edit codes that are associated with the field. For more information about edit codes, see your IBM documentation.

TIMESTAMP
combines a date and time and adds an optional microsecond to make a seven part value of the format *yyyy-mm-dd-hh.mm.ss[.nnnnnn]*. For example, a timestamp for precisely 2:25 p.m. on January 25, 1991, would be 1991-01-25-14.25.00.000000. Values in a column of this type have the same ranges as described earlier for DATE and TIME.

For more information about SQL data types, datetime formats, and edit codes that are used in the United States and other countries, see your IBM documentation.

DB2 Null and Default Values

DB2 has a special value called NULL. A DB2 NULL value means an absence of information and is analogous to a SAS missing value. When SAS/ACCESS reads a DB2 NULL value, it interprets it as a SAS missing value.

You can define a column in a DB2 table so that it requires data. To do this in SQL, you specify a column as NOT NULL. NOT NULL tells SQL to only allow a row to be added to a table if there is a value for the field. For example, NOT NULL assigned to the field CUSTOMER in the table SASDEMO.CUSTOMER does not allow a row to be added unless there is a value for CUSTOMER. When creating a DB2 table with SAS/ACCESS, you can use the DBNULL= data set option to indicate whether NULL is a valid value for specified columns.

DB2 columns can also be defined as NOT NULL WITH DEFAULT. For more information about using the NOT NULL WITH DEFAULT value, see your DB2 SQL reference manual.

Knowing whether a DB2 column allows NULLs, or whether the host system supplies a default value for a column that is defined as NOT NULL WITH DEFAULT, 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, it allows NULL values.

For more information about how SAS handles NULL values, see in *SAS/ACCESS for Relational Databases: Reference*.

Note: To control how SAS missing character values are handled by DB2, use the NULLCHAR= and NULLCHARVAL= data set options. △

LIBNAME Statement Data Conversions

The following table shows the default SAS variable formats that SAS/ACCESS assigns to DB2 data types during input operations when you use the LIBNAME statement .

Table 1.4 LIBNAME Statement: Default SAS Formats for DB2 Data Types

DB2 Data Type	SAS Data Type	Default SAS Format
CHAR(*n*)	character	$*n.*
VARCHAR(*n*)	character	$*n.*
LONG VARCHAR	character	$*n.*

DB2 Data Type	SAS Data Type	Default SAS Format
GRAPHIC(*n*), VARGRAPHIC(*n*), LONG VARGRAPHIC	character	$*n*.
INTEGER	numeric	11.
SMALLINT	numeric	6.
BIGINT	numeric	20.
DECIMAL	numeric	*m.n*
NUMERIC	numeric	*m.n*
FLOAT	numeric	none
DOUBLE	numeric	none
TIME	numeric	TIME8.
DATE	numeric	DATE9.
TIMESTAMP	numeric	DATETIME*m.n*

* *n* in DB2 data types is equivalent to *w* in SAS formats.

The following table shows the default DB2 data types that SAS/ACCESS assigns to SAS variable formats during output operations when you use the LIBNAME statement.

Table 1.5 LIBNAME Statement: Default DB2 Data Types for SAS Variable Formats

SAS Variable Format	DB2 Data Type
m.n	DECIMAL (*m,n*)
other numerics	DOUBLE
$*n*.	VARCHAR(*n*) (*n*<=4000) LONG VARCHAR(*n*) (*n*>4000)
datetime formats	TIMESTAMP
date formats	DATE
time formats	TIME

* *n* in DB2 data types is equivalent to *w* in SAS formats.

DBLOAD Procedure Data Conversions

The following table shows the default DB2 data types that SAS/ACCESS assigns to SAS variable formats when you use the DBLOAD procedure.

Table 1.6 PROC DBLOAD: Default DB2 Data Types for SAS Variable Formats

SAS Variable Format	DB2 Data Type
$*w*.	CHAR(*n*)
w.	DECIMAL(*p*)
w.d	DECIMAL(*p,s*)
IB*w.d*, PIB*w.d*	INTEGER

SAS Variable Format	DB2 Data Type
all other numerics*	DOUBLE
datetime*w.d*	TIMESTAMP
date*w.*	DATE
time.**	TIME

* Includes all SAS numeric formats, such as BINARY8 and E10.0.
** Includes all SAS time formats, such as TOD*w,d* and HHMM*w,d*.