CHAPTER 1

SAS/ACCESS for DB2 under z/OS

Introduction to the SAS/ACCESS Interface to DB2 under z/OS

This document describes the SAS/ACCESS interface to DB2 under z/OS. It includes details *only* about the SAS/ACCESS interface to DB2 under z/OS, and it should be used as a supplement to the main SAS/ACCESS documentation, *SAS/ACCESS for Relational Databases: Reference*.

Note: z/OS is the successor to the OS/390 operating system. SAS 9.1 for z/OS is supported on both OS/390 and z/OS operating systems and, throughout this document, any reference to z/OS also applies to OS/390, unless otherwise stated. △

LIBNAME Statement Specifics for DB2 under z/OS

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

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 z/OS.

connection-options
provides connection information and control which database you are connecting to. The connection options for the interface to DB2 under z/OS are the following:

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 is specified in the SAS system option, DB2SSID=. The DB2 subsystem ID is limited to four characters. See "Setting Your DB2 Subsystem Identifier" on page 25 for more information.

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 connect to only 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= LIBNAME option. There is no default value for this option.

For information about accessing a database server on Linux, UNIX, or Windows using a libref, see the REMOTE_DBTYPE= LIBNAME option.

Note: Refer to the installation instructions for this interface for information about configuring SAS to use the SERVER= option. △

LIBNAME-options
defines 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 z/OS, and presents default values where applicable. See the section about the SAS/ACCESS LIBNAME statement in *SAS/ACCESS for Relational Databases: Reference* for detailed information about these options.

Table 1.1 SAS/ACCESS LIBNAME Options

Option	Default Value
ACCESS=	none
AUTHID=	none
CONNECTION=	SHAREDREAD
CONNECTION_GROUP=	none
DBCOMMIT=	1000 when inserting rows; 0 when updating rows
DBCONINIT=	none
DBCONTERM=	none
DBCREATE_TABLE_OPTS=	none
DBGEN_NAME=	DBMS
DBLIBINIT=	none
DBLIBTERM=	none
DBNULLKEYS=	YES
DBSASLABEL=	COMPAT
DBSLICEPARM=	THREADED_APPS,2
DEFER=	NO
DEGREE=	ANY
DIRECT_EXE=	none
DIRECT_SQL=	YES
IN=	none
LOCATION=	none
MULTI_DATASRC_OPT=	NONE
PRESERVE_COL_NAMES=	NO
PRESERVE_TAB_NAMES=	NO
READ_ISOLATION_LEVEL=	DB2 z/OS determines the isolation level
READ_LOCK_TYPE=	DB2 z/OS handles locking
REMOTE_DBTYPE=	ZOS
REREAD_EXPOSURE=	NO
SCHEMA=	your user ID
SPOOL=	YES
UPDATE_ISOLATION_LEVEL=	DB2 z/OS determines the isolation level

Option	**Default Value**
UPDATE_LOCK_TYPE=	DB2 z/OS handles locking
UTILCONN_TRANSIENT=	NO

DB2 under z/OS LIBNAME Statement Example

In the following example, the libref MYLIB uses the DB2 under z/OS interface to connect to the DB2 database that is specified by the SSID= option, with a connection to the **`testserver`** remote server.

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

Data Set Options for DB2 under z/OS

The following list includes all of the SAS/ACCESS data set options that are supported for the DB2 under z/OS interface, except for the data set options that are available with the bulk load facility. Default values are provided where applicable. See the section on 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 z/OS

Option	Default Value
AUTHID=	current LIBNAME option setting
BULKLOAD=	NO
DBCOMMIT=	current LIBNAME option setting
DBCONDITION=	none
DBCREATE_TABLE_OPTS=	current LIBNAME option setting
DBFORCE=	NO
DBGEN_NAME=	DBMS
DBKEY=	none
DBLABEL=	NO
DBMASTER=	none
DBNULL=	YES
DBNULLKEYS=	current LIBNAME option setting
DBSASLABEL=	COMPAT
DBSLICE=	none
DBSLICEPARM=	THREADED_APPS,2
DBTYPE=	none

Option	Default Value
ERRLIMIT=	1
IN=	current LIBNAME option setting
LOCATION=	current LIBNAME option setting
NULLCHAR=	SAS
NULLCHARVAL=	a blank character
PRESERVE_COL_NAMES=	current LIBNAME option setting
READ_ISOLATION_LEVEL=	current LIBNAME option setting
READ_LOCK_TYPE=	current LIBNAME option setting
TRAP_151=	NO
UPDATE_ISOLATION_LEVEL=	current LIBNAME option setting
UPDATE_LOCK_TYPE=	current LIBNAME option setting

Pass-Through Facility Specifics for DB2 under z/OS

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 z/OS are as follows:

- The *dbms-name* is **DB2**.
- The CONNECT statement is optional.
- The interface supports connections to multiple databases.
- The CONNECT statement *database-connection-arguments* are as follows:

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 is specified in the SAS system option, DB2SSID=. The DB2 subsystem ID is limited to four characters. See "Setting Your DB2 Subsystem Identifier" on page 25 for more information.

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 connect to only 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= LIBNAME option. There is no default value for this option.

Note: Refer to the installation instructions for this interface for information about setting up DB2 z/OS to enable SAS to connect to the DRDA server when the SERVER= option is used. △

Examples

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 DB2SSID=, and performs a query (shown in highlighting) 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 following example creates the 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;
```

Autopartitioning Scheme for DB2 under z/OS

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

Overview

Autopartitioning for SAS/ACCESS to DB2 OS/390 and z/OS is a modulo (MOD) method, as introduced in the section on autopartitioning techniques in *SAS/ACCESS for Relational Databases: Reference*.

Threaded reads for DB2 on OS/390 and z/OS involve a trade-off. A threaded read with even distribution of rows across the threads substantially reduces elapsed time for your SAS step. Simply put, your job completes in less time. This is positive for job turnaround time, particularly if your job needs to complete within a constrained time window. However, threaded reads always increase the CPU time of your SAS job. They also increase the workload on DB2. If increasing CPU consumption or increasing DB2 workload are unacceptable, threaded reads can be turned off by specifying the SAS option DBSLICEPARM=NONE. To turn off threaded reads for all SAS jobs, assert DBSLICEPARM=NONE in the SAS restricted options table.

Autopartitioning Restrictions

SAS/ACCESS to DB2 under z/OS restricts which columns can be selected as an autopartitioning column. The following column types are eligible for partitioning:

- INTEGER
- SMALLINT
- DECIMAL.

Eligible DECIMAL columns must be confined to an integer range; specifically, DECIMAL columns with precision less than 10. For example, DECIMAL(5,0) and DECIMAL(9,2) are eligible.

Column Selection for MOD Partitioning

If multiple columns are eligible for partitioning, the engine queries the DB2 system tables for information about identity columns and simple indexes. Based on the information about the identity columns, simple indexes, column types, and column nullability, the partitioning column is selected according to the following priority scheme:

- Identity column
- Unique simple index: SHORT or INT, integral DECIMAL, and then non-integral DECIMAL.
- Non-unique simple index: SHORT or INT (NOT NULL), integral DECIMAL (NOT NULL), and then non-integral DECIMAL (NOT NULL).
- Non-unique simple index: SHORT or INT (nullable), integral DECIMAL (nullable), and then non-integral DECIMAL (nullable).
- SHORT or INT (NOT NULL), integral DECIMAL (NOT NULL), and then non-integral DECIMAL (NOT NULL).
- SHORT or INT (nullable), integral DECIMAL (nullable), and then non-integral DECIMAL (nullable)

If a nullable column is selected for autopartitioning, or *column-name* IS NULL is appended to the SQL generated for one read thread, ensuring NULL values are included in the result set.

How WHERE Clauses Restrict Autopartitioning

Autopartitioning does not use columns that appear in a SAS WHERE clause. For instance, the following DATA step cannot use autopartitioning since all the numeric columns in the table 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 you use autopartitioning, SAS/ACCESS to DB2 under z/OS defaults to two threads.

Using DBSLICE=

You can guarantee best performance for threaded reads by specifying a DB2-specific DBSLICE= table option.

Temporary Table Support for DB2 under z/OS

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 enables a single connection to be used across SAS DATA step and procedure boundaries as well as be shared between both the LIBNAME statement 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 TEMPORARY TABLE statement with the ON COMMIT PRESERVE 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.

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

```
libname temp db2 connection=global;

proc sql;
   connect to db2 (connection=global);
   exec (declare global temporary table temptab1
         like other.table on commit PRESERVE rows) 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 singe 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 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 'my.sas.library';
libname dept db2 connection=global schema=dschema;
libname temp db2 connection=global schema=SESSION;
/* Note that temporary table has a schema of SESSION */

proc sql;
   connect to db2 (connection=global);
   exec (declare global temporary table temptab1
        (dname char(20), deptno int)
         on commit PRESERVE rows) by db2;
quit;
```

In 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 append base=temp.temptab1 data=saslib.joindata;
run;
proc sql;
   connect to db2 (connection=global);
   select * from dept.deptinfo info, temp.temptab1 tab
        where info.deptno = tab.deptno;
   /* remove the rows for the next example */
   exec(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.

```
proc append base=temp.temptab1 data=saslib.transdat;
run;

proc sql;
   connect to db2 (connection=global);
   exec(update dschema.deptinfo d set deptno = (select deptn from
        session.temptab1)
        where d.dname = (select dname from session.temptab1)) by db2;
quit;
```

Calling Stored Procedures in DB2 under z/OS

Overview

A stored procedure is one or more SQL statements or supported 3GL language (such as C) statements that are compiled into a single procedure that resides in DB2. Stored procedures might contain static SQL — SQL statements that are hardwired, or static, in the program. Static SQL is optimized better for some DBMS operations, and in a carefully managed DBMS environment, it enables the programmer and the DBA to know the exact SQL that will be executed.

SAS is a usually a dynamic SQL generator. Stored procedure support, however, allows the DBA to encode static SQL in a stored procedure, and restrict SAS users to a tightly controlled interface.

SAS/ACCESS support for stored procedure include passing input parameters, retrieving output parameters into SAS macro variables, and retrieving result sets into SAS tables. Stored procedures can be called only from PROC SQL.

Examples

Basic Stored Procedure Call

Use CALL statement syntax to call a stored procedure:

```
CALL STORED_PROC
```

The simplest way to call a stored procedure is to use the EXECUTE statement in PROC SQL. In the following example, STORED_PROC is executed using a CALL statement. Any result set is not captured by SAS:

```
proc sql;
connect to db2;
execute (CALL STORED_PROC);
quit;
```

Stored Procedure That Returns a Result Set

You can also return the result to a SAS table. In the following example, STORED_PROC is executed using a CALL statement. The results are returned to a SAS table called SasResults:

```
proc sql;
connect to db2;
create table sasresults as select * from connection to db2 (call STORED_PROC);
quit;
```

Stored Procedure That Passes Parameters

The CALL statement syntax supports the passing of parameters. Input parameters can be specified as numeric constants, character constants, or a null value. Input parameters may also be passed via SAS macro variable references. To capture the value of an output parameter, a SAS macro variable reference is required. In the following example, a constant (1), an input/output parameter (:INOUT), and an output parameter

(:OUT) are used. The result set is returned to the SAS results table and the parameter outputs are captured in SAS macro variables INOUT and OUT.

```
proc sql;
connect to db2;
%let INOUT=2;
create table sasresults as select * from connection to db2
   (call STORED_PROC (1,:INOUT,:OUT));
quit;
```

Stored Procedure That Passes NULL Parameter

The following functions cause NULL to be passed as the parameter to the DB2 stored procedure:

Null string literals in the call:

```
CALL PROC('');
CALL PROC("")
```

Literal period or literal NULL in the call:

```
CALL PROC(.)
CALL PROC(NULL)
```

SAS macro variable set to NULL string:

```
%let charparm=;
CALL PROC(:charparm)
```

SAS macro variable set to period (SAS numeric value missing):

```
 %let numparm=.;
CALL PROC(:numparm)
```

Only literal period and literal NULL work generically for both DB2 character parms and DB2 numeric parms. For example, "" would be rejected for a DB2 numeric parm and

```
%let numparm=.;
```

would not pass a DB2 NULL for a DB2 character parm. As a literal, a period passes NULL for both numeric and character parms, but contained in a SAS macro variable it only constitutes a NULL for a DB2 numeric parm.

Passing NULL parms by simply omitting the argument is not supported. For example, you cannot use the following call in order to pass three NULL parms:

```
CALL PROC(,,)
```

Instead you would want to use the following:

```
CALL PROC(NULL,NULL,NULL)
```

Specifying the schema for a stored procedure

Use standard CALL statement syntax to execute a stored procedure that resides in another schema. For instance:

```
proc sql;
   connect to db2;
   execute (CALL OTHERSCHEMA.STORED_PROC);
quit;
```

If the schema is in mixed case or lower case, enclose the schema name in double quotation marks:

```
proc sql;
   connect to db2;
   execute (CALL "lowschema".STORED_PROC);
quit;
```

Executing remote stored procedures

If the stored procedure resides on a different DB2 instance, specify it with a valid three-part name:

```
proc sql;
   connect to db2;
   create table sasresults as select * from connection to db2
          (call OTHERDB2.PROCSCHEMA.PROC5 (1, NULL));
quit;
```

ACCESS Procedure Specifics for DB2 under z/OS

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

The DB2 z/OS interface supports all of the ACCESS procedure statements in interactive line, noninteractive, and batch modes. The ACCESS procedure specifics for DB2 under z/OS are as follows:

- □ The DBMS= value is **`db2`**.
- □ The *database-description-statements* are as follows:

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 is specified in the SAS system option, DB2SSID=. The DB2 subsystem ID is limited to four characters. See “Setting Your DB2 Subsystem Identifier” on page 25 for more information.

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 connect to only 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= LIBNAME option. There is no default value for this option.

Note: Refer to the installation instructions for this interface for information about configuring SAS to use the SERVER= option. △

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

In the DB2 z/OS 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.

- The TABLE= statement is as follows:

  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 z/OS) user ID. In batch mode, however, you must specify an authorization ID, otherwise an error message is generated.

Examples

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 following 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 Specifics for DB2 under z/OS

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

The DB2 under z/OS interface supports all of the DBLOAD procedure statements in interactive line, noninteractive, and batch modes. The DBLOAD procedure specifics for DB2 under z/OS are as follows:

- The DBMS= value is **`DB2`**.
- The database description statements are as follows:

  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 is specified in the SAS system option, DB2SSID=. The DB2 subsystem ID is limited to four characters. See “Setting Your DB2 Subsystem Identifier” on page 25 for more information.

  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 connect to only 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= LIBNAME option. There is no default value for this option.

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

  IN *database.tablespace* |’DATABASE *database*’
  : specifies the name 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 quotation marks.

- 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.

  Refer to "DB2 Null and Default Values" on page 29 for DB2-specific information about NULL values.

- The TABLE= statement is as follows:

  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 z/OS) user ID. In batch mode, however, you must specify an authorization ID, otherwise an error message is generated.

Examples

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 following 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;
```

Passing Joins to DB2 under z/OS

Multiple libref joins are passed down to the z/OS with the following exceptions:

- □ If the SERVER= option is specified for one of the librefs, it must also be specified for the others, and its value must be the same for all the librefs.
- □ If the LOCATION= option is specified for one of the librefs, its value must refer to the location of the DB2 subsystem that the other librefs point to, in case these do not specify the LOCATION= option. It must also have the same value as the other librefs where the option is specified.
- □ If the DIRECT_SQL= option is specified for one or multiple librefs, it must not be set to NO, NONE, or NOGENSQL.

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.

The DB2EXT Procedure

Overview

The DB2EXT procedure creates SAS data sets from DB2 under z/OS data. PROC DB2EXT runs interactively, noninteractively, and in batch mode. The generated data sets are not password protected. However, you can edit the saved code to add password protection.

PROC DB2EXT ensures that all SAS names that are generated from DB2 column values are unique. A numeric value is appended to the end of a duplicate name. If necessary, the procedure truncates the name when appending the numeric value.

Syntax

The syntax for the DB2EXT procedure is as follows:

PROC DB2EXT <*options*>;

FMT *column-number-1='SAS-format-name-1'*
<... *column-number-n='SAS-format-name-n'*>;

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

SELECT *DB2-SQL-statement*;

EXIT;

PROC DB2EXT Statement Options

IN=*SAS-data-set*

specifies a mapping data set that contains information such as DB2 names, SAS variable names, and formats for input to PROC DB2EXT.

Note: This option is available for use only with previously created mapping data sets. You cannot create new mapping data sets with DB2EXT. △

OUT=*SAS-data-set* | *libref.SAS-data-set*
specifies the name of the SAS data set that is created. If you omit OUT=, the data set is named "work.DATAn", where n is a number that is sequentially updated. The data set is not saved when your SAS session ends. If a file with the name that you specify in the OUT= option already exists, it is overwritten. However, you will be given a warning that this is going to happen.

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

Note: The subsystem name defaults to the subsystem defined by the option DB2SSID=. It only defaults to DB2 if neither the SSID= option or the DB2SSID= option are specified. △

FMT Statement

FMT *column-number-1='SAS-format-name-1'*
<... *column-number-n='SAS-format-name-n'*>;

The FMT statement assigns a SAS output format to the DB2 column that is specified by *column-number*. The *column-number* is determined by the order in which you list the columns in your SELECT statement. If you use SELECT *, the *column-number* is determined by the order of the columns in the database.

You must enclose the format name in single quotation marks. You can specify multiple column formats in a single FMT statement.

RENAME Statement

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

The RENAME statement assigns the *SAS-name* to the DB2 column that is specified by *column-number*. The *column-number* is determined by the order in which you list the columns in your SELECT statement. If you use SELECT *, the *column-number* is determined by the order of the columns in the database.

You can rename multiple columns in a single RENAME statement.

SELECT Statement

SELECT *DB2-SQL-statement*;

The *DB2-SQL-statement* defines the DB2 data that you want to include in the SAS data set. You can specify table names, column names, and data subsets in your SELECT statement. For example, the following statement selects all of the columns from the Employee table and includes only employees whose salary is greater than $40,000:

```
select * from employee where salary > 40000;
```

EXIT Statement

EXIT;

The EXIT statement terminates the procedure without further processing.

Examples

The following code creates a SAS data set named MyLib.NoFmt that includes three columns from the DB2 table EmplInfo. The RENAME statement changes the name of the third column that is listed in the SELECT statement (from **firstname** in the DB2 table to **fname** in the SAS data set.

```
/* specify the SAS library where the SAS data set will be saved */

libname mylib 'userid.xxx';

proc db2ext ssid=db25 out=mylib.nofmt;
   select employee, lastname, firstname from sasdemo.emplinfo;
   rename 3=fname;
run;
```

The following code uses a mapping file to specify which data to include in the SAS data set and how to format that data.

```
/* specify the SAS library where the SAS data set will be saved */
libname mylib 'userid.xxx';

/* specify the SAS library that contains the mapping data set */
libname inlib 'userid.maps';

proc db2ext in=inlib.mapping out=mylib.mapout ssid=db25;
run;
```

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 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 functions:

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 22 for more information about the types of modifications that are available and how they are used. Refer to "PROC DB2UTIL Example" on page 23 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*>;

RESET ALL|*SAS-name*| COLS;

SQL *SQL-statement*;

UPDATE;

WHERE *SQL-WHERE-clause*;

ERRLIMIT=*error-limit*;

EXIT;

PROC DB2UTIL Statements and 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 22 for a detailed description of this option. FUNCTION= is required unless you specify an SQL file with the SQLIN= option.

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 z/OS sequential data set or a member of a z/OS 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. See "Setting Your DB2 Subsystem Identifier" on page 25 for more information.

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

You can insert observations from a SAS data set into a DB2 table as rows in the table. To use this 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. However, all DB2 columns must be mapped to a SAS column.

Updating Data

You can 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

You can 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.

PROC DB2UTIL Example

The following example uses the UPDATE function in PROC DB2UTIL 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 is updated from the 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 is not found in the Testid.Employees table and therefore is not updated. The warning that appears in the SAS log can be ignored.

SAS System Options and Settings for DB2 under z/OS

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

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 this query.

```
libname mylib db2 ssid=db2;

proc print data=mylib.staff;
run;

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

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=*decimal-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 SAS.

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 DB2IN= 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 quotation marks.

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.

DB2RRS | NODB2RRS

specifies the attachment facility to be used for a 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 "How the Interface to DB2 Works" on page 48.

DB2RRSMP | NODB2RRSMP

specifies that the multi-phase commit and rollback calls, SRRCMIT and SRRBACK, are used instead of the COMMIT and ROLLBACK SQL statements. This option is ignored unless DB2RRS is specified. This option is available only at invocation.

Specify NODB2RRSMP, the default, when DB2 is the only Resource Manager for your application. Specify DB2RRSMP when your application has other resource managers, which requires the use of the multi-phase calls. Using the multi-phase calls when DB2 is your only resource manager can have performance implications. Using COMMIT and ROLLBACK when you have more than one resource manager can result in an error, depending upon the release of DB2.

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. For more information, see "Setting Your DB2 Subsystem Identifier" on page 25.

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

You can override the DB2SSID= system option with the SSID= connection 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 SAS. This option does not affect the Pass-Through facility, PROC DBLOAD, or the VS compatibility procedures.

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.
- □ 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= connection 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.

Naming Conventions for DB2 under z/OS

The PRESERVE_COL_NAMES= and PRESERVE_TAB_NAMES= options determine how the interface to DB2 under z/OS handles case-sensitivity, spaces, and special characters. The default for both of these options is NO. DB2 converts table and column names to uppercase by default. However, DB2 is a case-sensitive DBMS. To preserve the case of the table and column names that you send to DB2, use quotation marks around the names. See the section about the LIBNAME statement in *SAS/ACCESS for Relational Databases: Reference* for additional information about these options.

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 quotation marks, it can start with and contain any character. Depending on how your string delimiter is set, quoted strings can contain quotation marks 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 quotation marks, 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.

Data Types for DB2 under z/OS

Every column in a table has a name and a data type. The SAS/ACCESS interface to DB2 handles all DB2 data types.

String Data

The following are the DB2 string data types:

CHAR(*n*)
: specifies a fixed-length column of length *n* for character string data. The maximum for *n* is 255.

VARCHAR(*n*)
: specifies a varying-length column for character string data. *n* specifies the maximum length of the string. If *n* is greater than 255, 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 following are the DB2 numeric data types:

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 $0 \leq s \leq p$.

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 might contain data values that are out of range for SAS, 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 Null and Default Values

DB2 has a special value that is 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.

DB2 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. 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. 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 1.3 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.

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 LIBNAME statement input operations.

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

DB2 Column Type	Default SAS Format
CHAR(*n*), VARCHAR(*n*), LONG VARCHAR(*n*)	$*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 SAS/ACCESS assigns to SAS variable formats during output operations.

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

SAS Variable Format	DB2 Data Type
$*w*., $CHAR*w*., $VARYING*w*., $HEX*w*.	CHARACTER(*w*) for 1–255 VARCHAR(*w*) for >255
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 variable formats that SAS/ACCESS assigns to DB2 data types when you use the ACCESS procedure.

Table 1.6 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)

DB2 Column Type	Default SAS Format
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 SAS/ACCESS assigns to SAS variable formats when you use the DBLOAD procedure.

Table 1.7 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
any datetime format	TIMESTAMP
all other numeric formats	FLOAT

Maximizing DB2 under z/OS 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 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, or you can use the DB2 under z/OS interface's bulk load facility .

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 to the SAS log the dynamic SQL that is generated by the DB2 engine and all other SQL that is executed by the DB2 engine. 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 steps 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 might 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 do not use the DBKEY= option, the entire table is retrieved into SAS, and the join is performed in SAS.
- 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.

Optimizing 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 enables 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 that are opened for input fetch from cursors do not issue commits, while update openings might, and output openings 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 openings get their own connections. CONNECTION=SHAREDREAD allows for 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 usually 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 openings, regardless of which libref you use, as opposed to one connection per libref for input openings. In a single-user environment (as opposed to a server session), you might know that you will not have multiple openings 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 opening 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 "DB2 under z/OS Information for the Database Administrator" on page 47, 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 opening use the first connection. UNIQUE causes each opening to have its own connection. If you only have one opening 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 opening.

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 opening 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.

Passing SAS Functions to DB2 under z/OS

The interface to DB2 under z/OS passes the following SAS SQL functions to DB2 under z/OS for processing. Where the DB2 function name is different than the SAS function name, the DB2 name appears in parentheses. See the section about optimizing PROC SQL in *SAS/ACCESS for Relational Databases: Reference* for more information.

ABS

ARCOS (ACOS)

ARSIN (ASIN)

ATAN

AVG

CEIL

COS

COSH

EXP

FLOOR

LOWCASE (LCASE)

LOG

LOG10

MAX

MIN

MOD

SIGN

SIN

SINH

SQRT

TAN

TANH

UPCASE (UCASE)

SUM

COUNT

Note: None of these functions exist in DB2 prior to DB2 V6, so these functions are not passed to the DBMS in DB2 V5. These functions are also not passed to the DBMS when you connect using DRDA, because there is no way to determine what location you are connected to and which functions are supported there. △

The following functions *are* passed to the DBMS in DB2 V5, as well as DB2 V6 and later. These functions are not passed to the DBMS when you connect using DRDA.

YEAR

MONTH

DAY

Locks in the DB2 under z/OS Interface

The following LIBNAME and data set options enable you to control how the interface to DB2 under z/OS handles locking and isolation levels. For more information, see your DB2 documentation. See the section about the LIBNAME statement in *SAS/ACCESS for Relational Databases: Reference* for additional information about these options.

READ_LOCK_TYPE=TABLE

UPDATE_LOCK_TYPE=TABLE

READ_ISOLATION_LEVEL= CS | UR | RR | "RR KEEP UPDATE LOCKS" | RS | "RS KEEP UPDATE LOCKS"
: The valid values for this option are described in the following table. The default isolation level is determined by DB2.

Table 1.8 Isolation Levels for DB2 under z/OS

Value	Isolation Level
CS	Cursor stability
UR	Uncommitted read
RR	Repeatable read
RR KEEP UPDATE LOCKS*	Repeatable read keep update locks
RS	Read stability
RS KEEP UPDATE LOCKS*	Read stability keep update locks

* When specifying a value that consists of multiple words, enclose the entire string in quotation marks.

UPDATE_ISOLATION_LEVEL= CS | UR | RR | "RR KEEP UPDATE LOCKS" | RS | "RS KEEP UPDATE LOCKS"
The valid values for this option are described in the preceding table. The default isolation level is determined by DB2.

DB2 under z/OS Bulk Loading

By default, the DB2 under z/OS interface loads data into tables by preparing an SQL INSERT statement, executing the INSERT statement for each row, and issuing a COMMIT. If you specify BULKLOAD=YES, the DB2 LOAD utility is invoked, enabling you to bulk load the rows of data as a single unit, which can significantly enhance performance. For smaller tables, the extra overhead of the bulk load process might slow performance. For larger tables, the speed of the bulk load process outweighs the overhead costs.

Note: When using bulk load, consult the SYSPRINT output for information about the load. If you run the LOAD utility and it fails, ignore the messages in the SAS log because they might be inaccurate. However, if there were errors prior to running the LOAD utility, the error messages in the SAS log might be valid. △

In the SAS/ACCESS interface to DB2 under z/OS, bulk loading capability is provided via DSNUTILS, which is an IBM stored procedure that invokes the DB2 LOAD utility. DSNUTILS is included in DB2 Version 6 and later, and it is available for DB2 Version 5 in a maintenance release. Because the LOAD utility is complex, familiarize yourself with it before you use it through SAS/ACCESS. Also, check with your database administrator to determine whether this utility is available.

Data Set Options for Bulk Loading

The bulk loading data set options for DB2 under z/OS all begin with BL_ (for bulk load) and are listed below. You must specify BULKLOAD=YES to use any of the BL_ data set options; if BULKLOAD=YES is not specified, all of the BL_ options are ignored. (The DB2 under z/OS interface alias for BULKLOAD= is DB2LDUTIL=.)

BL_DB2CURSOR='*string*'
specifies a string that contains a valid DB2 SELECT statement that points to either local or remote objects (tables or views). It has no default value and is used only if you specify it. You can use it to load DB2 tables directly from other DB2

and non-DB2 objects. However, before you can select data from a remote location, your database administrator must first populate the communication database with the appropriate entries.

BL_DB2DEVT_PERM= *unit-specification*
: specifies the unit address or generic device type that is used for the permanent data sets that are created by the LOAD utility, as well as SYSIN, SYSREC, and SYSPRINT when allocated by SAS. The default is SYSDA.

BL_DB2DEVT_TEMP= *unit-specification*
: specifies the unit address or generic device type that is used for the temporary data sets that are created by the LOAD utility (Pnch, Copy1, Copy2, RCpy1, RCpy2, Work1, Work2). The default is SysDa.

BL_DB2DISC= *data-set-name*
: specifies the SYSDISC data set name for the LOAD utility and is allocated by the DSNUTILS procedure with DISP=(NEW,CATLG,CATLG). This must be the name of a nonexistent data set (except on a RESTART, because it would have been created already). The utility allocates it as DISP=(MOD,CATLG,CATLG) on a RESTART. The default is a generated data set name (which appears in the output that is written to the DB2PRINT location).

BL_DB2ERR= *data-set-name*
: specifies the SYSERR data set name for the LOAD utility and is allocated by the DSNUTILS procedure with DISP=(NEW,CATLG,CATLG). This must be the name of a nonexistent data set (except on a RESTART, because it would have been created already). The utility allocates it as DISP=(MOD,CATLG,CATLG) on a RESTART. The default is a generated data set name (which appears in the output that is written to the DB2PRINT location).

BL_DB2IN= *data-set-name*
: specifies the SYSIN data set name for the LOAD utility and is allocated based on the value of BL_DB2LDEXT. When allocated new, SPACE=(trk,(10,1),rilse). The default is a generated data set name (which appears in the DB2PRINT output) with the following DCB attributes:

  DSORG= PS

  RECFM= VB

  LRECL= 516

  BLKSZE= 23476.

  The following are the supported DCB attributes for existing data sets:

  DSORG= PS

  RECFM= F, FB, FS, FBS, V, VB, VS, or VBS

  LRECL= any valid value for RECFM, which is < 32,760

  BLKSIZE= any valid value for RECFM, which is < 32,760.

BL_DB2LDCT1= '*string*'
: specifies a string containing segment of the Load Utility Control Statement between 'LOAD DATA' and 'INTO TABLE'. Valid control statement options include, but are not limited to, RESUME, REPLACE, LOG, ENFORCE. The default is nothing.

  The BL_DB2LDCT options enable you to specify sections of the control statement, which the engine incorporates into the control statement that it generates. (These options have no effect when BL_DB2LDEXT=USERUN.) You can use these options as an alternative to specifying BL_DB2LDEXT=GENONLY and then editing the control statement to include options that the engine cannot generate. In some cases, specifying at least one of these options is necessary, for example, when you run the utility against an existing table where either RESUME or REPLACE needs to be specified.

The LOAD utility requires that the control statement be in upper case (except for things like table or column names, which must match the table). Specify the values for the BL_DB2LDCT options in the correct case. The interface to DB2 under z/OS cannot convert the entire control statement to upper case, because it might contain table or column names that must remain in lower case.

BL_DB2LDCT2= '*string*'
specifies a string that contains a segment of the Load Utility Control Statement between INTO TABLE *table-name* and the (*field-specification*). Valid control statement options include, but are not limited to, PART, PREFORMAT, RESUME, REPLACE, WHEN. The default is nothing.

BL_DB2LDCT3= '*string*'
specifies a string that contains a segment of the Load Utility Control Statement after (*field-specification*). This option is currently unnecessary, but will handle any options that might be defined for this location in later versions of DB2. The default is nothing.

BL_DB2LDEXT= GENRUN | GENONLY | USERUN
specifies the mode of execution for the DB2 LOAD utility, which involves creating data sets that are needed by the utility and invoking the utility.

GENRUN
generates the control file (SYSIN) and the data file (SYSREC) and invokes the utility with them. This is the default.

GENONLY
generates the control file (SYSIN) and the data file (SYSREC) but does not invoke the utility. Use this method when you need to edit the control file or verify the generated control statement or data before running the utility.

USERUN
uses existing control and data files and runs the utility with them. The existing files can be from a previous run or from existing batch utility jobs you used in the past. Use this execution method when you restart a previously stopped invocation of the utility.

All valid data sets that the utility accepts are supported when BL_DB2LDEXT=USERUN. However, syntax errors from the utility can occur because no parsing is done when reading in the SYSIN data set. Specifically, neither embedded comments (beginning with a double dash '–') nor columns 73 through 80 of RECFM=FB LRECL=80 data sets are stripped out of the control statement. The solution is to remove embedded comments and columns 73 through 80 of RECFM=FB LRECL=80 data sets from the data set. (This is not an issue when using engine generated SYSIN data sets because those data sets are RECFM=VB and do not have embedded comments.)

BL_DB2MAP= *data-set-name*
specifies the SYSMAP data set name for the LOAD utility and is allocated by the DSNUTILS procedure with DISP=(NEW,CATLG,CATLG). This must be the name of a nonexistent data set (except on a RESTART, because it would have been created already). The utility allocates it as DISP=(MOD,CATLG,CATLG) on a RESTART. The default is a generated data set name (which appears in the output written to the DB2PRINT location).

BL_DB2PRINT= *data-set-name*

specifies the SYSPRINT data set name for the LOAD utility and is allocated with DISP=(NEW,CATLG,DELETE) and with SPACE=(trk,(10,1),rlse). The default is a generated data set name (which appears in the output written to the DB2PRINT dsn) with the following DCB attributes:

DSORG= PS

RECFM= VBA

LRECL= 258

BLKSIZE= 262 – 32760.

Also specify BL_DB2PRNLOG=, so that you see the generated data set name in the SAS log.

BL_DB2PRNLOG= YES | NO

determines whether the SYSPRINT output is written to the SAS log. The SYSPRINT output is always written to the data set name that is specified by BL_DB2PRINT=. This option lets you see the output in the SAS log as well.

YES

specifies that the SYSPRINT output is written to the SAS log. This is the default.

NO

specifies that the SYSPRINT output is not written to the SAS log.

BL_DB2REC= *data-set-name*

specifies the SYSREC data set name for the LOAD utility and is allocated based on the value of BL_DB2LDEXT. When allocated new, SPACE=(cyl,(BL_DB2RECSP, 10%(BL_DB2RECSP)),rlse). The default is a generated data set name (which appears in the output that is written to the DB2PRINT data set name). The supported DCB attributes for existing data sets are as follows:

DSORG= PS

RECFM= FB

LRECL= any valid value for RECFM

BLKSIZE= any valid value for RECFM.

BL_DB2RECSP= *primary-allocation*

determines the number of cylinders to specify as the primary allocation for the SYSREC data set, when it is created. The default is 10. The secondary allocation is 10% of the primary allocation.

BL_DB2RSTRT= NO | CURRENT | PHASE

tells the LOAD utility whether the current load is a restart and, for a restart, indicates where to begin. When you specify a value other than NO, you must also specify BL_DB2TBLXST=YES and BL_DB2LDEXT=USERUN.

NO

indicates a new invocation of the LOAD utility, not a restart. This is the default.

CURRENT

indicates to restart at the last commit point.

PHASE

indicates to restart at the beginning of the current phase.

BL_DB2SPC_PERM= *primary-allocation*

determines the number of cylinders to specify as the primary allocation for the permanent data sets that are created by the LOAD utility. The permanent data sets are Disc, Maps, and Err. The default is 10. The secondary allocation is controlled by the DSNUTILS procedure and is 10% of the primary allocation.

BL_DB2SPC_TEMP= *primary-allocation*

determines the number of cylinders to specify as the primary allocation for the temporary data sets that are created by the LOAD utility. The temporary data sets are Pnch, Copy1, Copy2, RCpy1, RCpy2, Work1, and Work2. The default is 10. The secondary allocation is controlled by the DSNUTILS procedure and is 10% of the primary allocation.

BL_DB2TBLXST= YES | NO

indicates whether the LOAD utility runs against an existing table.

YES

specifies that the LOAD utility runs against an existing table. This is *not* a replacement operation. See details below.

NO

specifies that the LOAD utility does not run against an existing table. This is the default.

At this time, SAS/ACCESS does not support table replacement. You cannot simply create a new copy of an existing table, replacing the original table. Instead, you must delete the table and then create a new version of it.

The DB2 LOAD utility does not create tables; it loads data into existing tables. The DB2 under z/OS interface creates a table before loading data into it (whether you use SQL INSERT statements or invoke the LOAD utility). You might want to invoke the utility for an existing table that was not created by the DB2 engine. In this case, specify BL_DB2TBLXST=YES to tell the engine that the table already exists. When this option is set to YES, the engine does not verify that the table does not already exist (which eliminates the NO REPLACE error), and the engine does not create the table. Since BULKLOAD= is not valid for update opens of tables, which include appending to an existing table, use BL_DB2TBLXST= with an output open (which would normally create the table) to accomplish appending, or to use the LOAD utility against a previously created table. Also, you can use BL_DB2TBLXST= in conjunction with BL_DB2LDEXT=GENONLY if the table does not yet exist and you do not want to create it or load it yet. In this case the control and data files are generated, but the table is neither created nor loaded.

Because the table might be empty or might contain rows, specify the appropriate LOAD utility control statement values for REPLACE and/or RESUME by using the BL_DB2LDCT1 and/or BL_DB2LDCT2 options.

The data to be loaded into the existing table must match the column types of the table. The engine does not try to verify the input data with the table definition. Any incompatible differences are flagged by the LOAD utility.

BL_DB2UTID= *utility-id*

specifies a unique identifier for a given run of the DB2 LOAD utility. This is a character string up to 16 bytes long. By default, the utility ID is the user ID concatenated with the second-level data set name qualifier. The generated ID appears in the output written to the DB2PRINT data set name. This name generation makes it easy to associate all of the information for each utility execution, and to separate it from other executions.

File Allocation and Naming for Bulk Loading

When you use bulk loading, the following files (data sets) are allocated:

- □ DB2's DSNUTILS allocates as new and catalogs the files SysDisc, SysMap, and SysErr, unless BL_DB2LDEXT= USERUN (in which case the data sets are allocated as old and are kept).
- □ The DB2 interface engine allocates as new and catalogs the files SysIn and SysRec when the execution method specifies to generate them.
- □ The DB2 interface engine allocates as new and catalogs the file SysPrint when the execution method specifies to run the utility.

All of the allocations of these data sets are reversed by the end of the step. If errors occur prior to generation of the SysRec, any of these data sets that were allocated as new and cataloged are deleted as part of cleanup (since they would be empty).

The interface engine uses the following options when it allocates nonexisting SYS data set names:

- □ BL_DB2DEVT_PERM= and BL_DB2SPC_PERM= are used by DSNUTILS for SysDisc, SysMap, and SysErr.
- □ BL_DB2DEVT_PERM= is also used by the engine for SysIn, SysRec, and SysPrint.
- □ BL_DB2RECSPC= is used for SysRec and is necessary because the engine cannot determine how much space the SysRec requires; it depends on the volume of data being loaded into the table.
- □ BL_DB2DEVT_TEMP= and BL_DB2SPC_TEMP= are used by DSNUTILs to allocate the other data set names required by the LOAD utility.

The following table shows how SysIn and SysRec are allocated based on the values of BL_DB2LDEXT= and BL_DB2IN=, and BL_DB2REC=.

Table 1.9 SysIn and SysRec Allocation

BL_DB2LDEXT=	BL_DB2IN=/ BL_DB2REC=	Data set name	DISPOSITION
GENRUN	not specified	generated	NEW, CATALOG, DELETE
GENRUN	specified	specified	NEW, CATALOG, DELETE
GENONLY	not specified	generated	NEW, CATALOG, DELETE
GENONLY	specified	specified	NEW, CATALOG, DELETE
USERUN	not specified	ERROR	
USERUN	specified	specified	OLD, KEEP, KEEP

When the interface to DB2 under z/OS uses existing files, you must specify the filenames. When the interface generates the files, it creates them with names you provide or with unique names it generates. Engine-generated filenames use system generated data set names with the format *SYSyyddd.Thhmmss.RA000.jobname.name.Hgg* where

SYSyyddd
: is replaced by the user ID. The user ID used to prequalify these generated data set names is determined the same as within the rest of SAS, except when running in a server environment, where the authenticated ID of the client is used.

name
is replaced by the given SYS ddname of the data set.

For example, if you do not specify any data set names and run GENRUN under TSO, you get a set of files allocated with names such as

```
USERID.T125547.RA000.USERID.DB2DISC.H01
USERID.T125547.RA000.USERID.DB2ERR.H01
USERID.T125547.RA000.USERID.DB2IN.H01
USERID.T125547.RA000.USERID.DB2MAP.H01
USERID.T125547.RA000.USERID.DB2PRINT.H01
USERID.T125547.RA000.USERID.DB2REC.H01
```

Because it produces unique names, even within a sysplex (within one second per user ID per system), this naming convention makes it easy to associate all of the information for each utility execution, and to separate it from other executions.

Examples

Use the following LIBNAME statements for all examples:

```
libname db2lib db2;
libname shlib db2 connection=shared;
```

Create a table.

```
data db2lib.table1 (bulkload=yes);
   x=1;
   name='Tom';
run;
```

Append Table1 to itself.

```
data shlib.table1
   (bulkload=yes bl_db2tblxst=yes bl_db2ldct1='RESUME YES');
   set shlib.table1;
run;
```

Replace Table1 with itself.

```
data shlib.table1
   (bulkload=yes bl_db2tblxst=yes bd_db2ldct1='REPLACE');
   set shlib.table1;
run;
```

Load DB2 tables directly from other objects.

```
data db2lib.emp (bulkload=yes);
   bl_db2ldct1='replace log no nocopypend' bl_db2cursor='select * from dsn8710.emp');
   set db2lib.emp (obs=0);
run;
```

You can also use this option in a PROC SQL statement to load DB2 tables directly from other objects, as shown below.

```
options sastrace=',,,d';
libname db2lib db2 authid=dsn8710;
libname mylib db2;

proc delete data mylib.emp;
run;
```

```
proc sql;
     connect to db2;
     create table mylib.emp
            (BULKLOAD=YES
             BL_DB2LDCT1='REPLACE LOG NO NOCOPYPEND'
             BL_DB2CURSOR='SELECT FIRSTNAME, LASTNAME, WORKDEPT,
                                  HIREDATE, JOB, SALARY, BONUS, COMM
                           FROM DSN8710.EMP')
     as select firstname, lastname, workdept,
               hiredate, job, salary, bonus, comm
        from db2lib.emp (obs=0);
quit;
```

Here is another, similar example.

```
options sastrace=',,,d';
libname db2lib db2 authid=dsn8710;
libname mylib db2;

proc delete data mylib.emp;
run;

proc sql;
     connect to db2;
     create table mylib.emp
            (BULKLOAD=YES
             BL_DB2LDCT1='REPLACE LOG NO NOCOPYPEND'
             BL_DB2CURSOR='SELECT FIRSTNAME, LASTNAME, WORKDEPT,
                                  HIREDATE, JOB, SALARY, BONUS, COMM
                           FROM DSN8710.EMP'
             BL_DB2LDCT3='RUNSTATS TABLESPACE DSNDB04.TEMPTTABL
                          TABLE(ALL) INDEX(ALL) REPORT YES')
     as select firstname, lastname, workdept,
               hiredate, job, salary, bonus, comm
        from db2lib.emp (obs=0);
quit;
```

Generate control and data files, create the table, but do not run the utility to load it.

```
data shlib.table2 (bulkload=yes
   bl_db2ldext=genonly bl_db2in='userid.sysin' bl_db2rec='userid.sysrec');
   set shlib.table1;
run;
```

Use the control and data files generated above to load the table. The OBS=1 data set option on the input file prevents the DATA step from reading the whole file. Because the data is really in SysRec, you need only the input file to satisfy the engine.

```
data db2lib.table2 (bulkload=yes bl_db2tblxst=yes
     bl_db2ldext=userun bl_db2in='userid.sysin' bl_db2rec='userid.sysrec');
   set db2lib.table1 (obs=1);
run;
```

A more efficient approach than the previous example is to eliminate going to DB2 to read even one observation from the input table. This also means that the DATA step processes only one observation, without any input I/O. Note that the one variable V is not on the table. Any variables listed here (there is no need for more than one), are irrelevant because the table already exists; they are not used.

```
data db2lib.table2 (bulkload=yes bl_db2tblxst=yes
     bl_db2ldext=userun bl_db2in='userid.sysin' bl_db2rec='userid.sysrec');
   v=0;
run;
```

Generate control and data files, but do not create the table or run the utility. Setting BL_DB2TBLXST=YES when the table does not exist prevents you from creating the table; this only makes sense because you are not going to load any data into the table at this time.

```
data db2lib.table3 (bulkload=yes bl_db2tblxst=yes
     bl_db2ldext=genonly bl_db2in='userid.sysin' bl_db2rec='userid.sysrec');
   set db2lib.table1;
run;
```

Use the control and data files generated in the preceding example to load the table. The OBS=1 data set option on the input file prevents the DATA step from reading the whole file. In this case, you must specify the input file because it contains the column definitions that are necessary to create the table.

```
data shlib.table3 (bulkload=yes bl_db2ldext=userun
     bl_db2in='userid.sysin' bl_db2rec='userid.sysrec');
   set shlib.table1 (obs=1);
run;
```

If you know the column names, a more efficient approach than the previous example is to eliminate going to DB2 to get the column definitions. In this case, the variable names and data types must match, because they are used to create the table. However, the values specified for the variables are not included on the table, because all of the data to load comes from the existing SysRec.

```
data db2lib.table3 (bulkload=yes bl_db2ldext=userun
     bl_db2in='userid.sysin' bl_db2rec='userid.sysrec');
   x=0;
   name='???';
run;
```

You can use other applications that do output processing.

```
data work.a;
   x=1;
run;

proc sql;
   create db2lib.table4 (bulkload=yes) as select * from a;
quit;
```

Understanding DB2 under z/OS Client/Server Authorization

When you use the SAS/ACCESS interface to DB2 under z/OS, you can enable each client to control its own connections using its own authority (instead of sharing connections with other clients) by using DB2's *Recoverable Resource Manager Services Attachment Facility* (RRSAF). See "DB2 Attachment Facilities (CAF and RRSAF)" on page 49 for information about this facility.

When you use the SAS/ACCESS interface to DB2 under z/OS with RRSAF, the authorization mechanism works differently than it does in Base SAS:

- □ In Base SAS, the SAS server *always* validates the client's authority before allowing the client to access a resource.
- □ In the SAS/ACCESS interface to DB2 under z/OS (with RRSAF), DB2 checks the authorization identifier that is carried by the connection from the SAS server. In most situations, this is the client's authorization identifier. In one situation, however, this is the SAS server's authorization identifier. A client can access a resource by using the *server's* authorization identifier only if the client uses a libref that was predefined in the server session.

In the following example, a user assigns the libref SRVPRELIB in the SRV1 server session. Then in the client session, a user issues a LIBNAME statement that makes a logical assignment using the libref MYPRELIB, and the user specifies the LIBNAME option SERVER=srv1. This enables the client to access resources by using the server's authority for the connection.

1 In the server session:

```
libname srvprelib db2 ssid=db25;
proc server id=srv1;
run;
```

2 In the client session:

```
libname myprelib server=srv1 slibref=srvprelib;
proc print data=myprelib.db2table;
run;
```

In the following example, because the client specifies a regular libref, MYDBLIB, the client has its own authority for the connections.

1 In the server session:

```
libname myprelib db2 ssid=db25;
proc server id=srv1;
run;
```

2 In the client session:

```
libname mydblib server=srv1 roptions='ssid=db25' rengine=db2;
proc print data=mydblib.db2table;
run;
```

In the following table, SAS/SHARE clients use LIBNAME statements to access SAS data libraries and DB2 data through the server. In this description, a *logical* LIBNAME statement is a statement that associates a libref with another, previously-assigned libref.

Table 1.10 Librefs and Their Authorization Implications

Client Session	

libname local v8 'SAS.data.library' disp=old;
libname dblocal db2 connection=unique;
	These statements execute in the client session. these are local assignments. The authority ID is the ID of the client.

libname remote 'SAS.data.library' server=serv1 rengine=v8 roptions='disp=old';
libname dbremote server=serv1 rengine=db2 roptions='connection=unique';
	These statements execute in the server session on behalf of the client. Libref Remote is a base engine remote assignment. Libref DbRemote is a DB2 engine remote assignment. In both cases, the authority ID is the ID of the client.

Server Session (id=serv1)	

libname predef v8 'SAS.data.library' disp=old;
libname dbpredef db2 connection=unique;
	Because librefs PreDef and DbPreDef are defined in the server session, they can only be referenced by a client using a logical LIBNAME statement. There is no authority ID because clients can not access these librefs directly.

Logical Assignments - Client Session	

libname alias (local);
libname dbalias (dblocal);
	These statements create aliases ALIAS and DBALIAS for librefs Local and DbLocal, which were assigned in the client session above. The authority ID is the ID of the client.

libname logic server=serv1 slibref=predef;
libname dblogic server=serv1 slibref=dbpredef;
	These statements refer to librefs PreDef and DbPreDef, which were assigned in the server session above.
	Libref Logic is a base engine logical assignment of remote libref PreDef. The authority ID for libref Logic is the ID of the client
	Libref DbLogic is a DB2 engine logical assignment of remote libref DbPreDef. The authority ID for libref DbLogic is the ID of the server.

For the base engine librefs Remote and Logic, the authority which is verified is that of the client (this is true for as for all Base SAS engine assignments). Although the DB2 engine librefs DbRemote and DbLogic refer to the same resources, the authority verified for DbRemote is that of the client, whereas the authority verified for DbLogic is that of the server. When using the SAS/ACCESS DB2 engine, you can determine whose authority (client or server) is used to access DB2 data.

Non-Libref Connections

When you make connections using the Pass-Through Facility or view descriptors, the connections to the database are not based on a DB2 engine libref. A connection that is created in the server, by using these features from a client, always has the authority of the client, because there is no server-established connection to reference.

The following example uses the SAS/SHARE Remote Pass-Through Facility. The client has its own authority for the connections.

1 In the server session:

```
proc server id=srv1;
run;
```

2 In the client session:

```
proc sql;
   connect to remote (server=srv1 dbms=db2 dbmsarg=(ssid=db25));
   select * from connection to remote
      (select * from db2table);
   disconnect from remote;
quit;
```

The following example uses a previously created view descriptor. The client has its own authority for the connections. Note that the preassigned libref PreLib and the client-assigned libref MyLib have no relevant difference, because these are base engine librefs and not DB2 engine librefs.

1 In the server session:

```
libname prelib V8 'SAS.data.library';
proc server id=srv1;
run;
```

2 In the client session:

```
libname prelib server=srv1;
proc print data=prelib.accview;
run;
```

3 In the client session:

```
libname mylib 'SAS.data.library2' server=srv1 rengine=v8;
proc print data=mylib.accview;
run;
```

Known Issues with RRSAF Support

SAS/SHARE can use various communication access methods to communicate with clients. You can specify these through the COMAMID and COMAUX1 system options.

When you use XMS (Cross Memory Services) as an access method, DB2 also uses XMS in the same address space. Predefining DB2 server librefs (prior to invoking PROC SERVER) can result in errors due to the loss of the XMS Authorization Index, because both SAS and DB2 are acquiring and releasing it. When using XMS as an access method, use only client-assigned librefs on the server.

This problem does not occur when you use the TCPIP access method. Therefore, if you use TCPIP instead of XMS, you can use both client-assigned (client authority) and server-preassigned (server authority) librefs. Also, you can use either access method if your connection is not base on a libref (client authority).

DB2 under z/OS Information for the Database Administrator

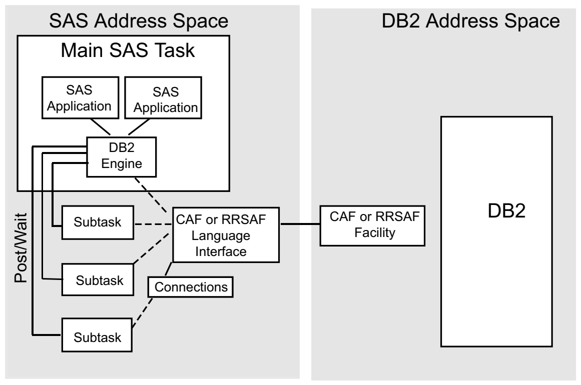
This section includes information about how the DB2 engine works, how SAS connects to DB2, and how the DB2 engine accesses DB2 system catalogs.

How the Interface to DB2 Works

The SAS/ACCESS interface to DB2 uses the either the Call Attachment Facility (CAF) or the Recoverable Resource Management Services Attachment Facility (RRSAF) to communicate with the local DB2 subsystem. Both attachment facilities enable programs to connect to DB2 and to use DB2 for SQL statements and commands. The interface to DB2 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. SAS and SAS executables run under one TCB, or task.

The DB2 LIBNAME statement enables SAS users to connect to DB2 more than once. Because the CAF and RRSAF allow only one connection per TCB, the interface to DB2 attaches a subtask for each subsequent connection that is initiated. It uses the ATTACH, DETACH, POST, and WAIT assembler macros to create and communicate with the subtasks. It does not limit the number of connections/subtasks that a single SAS user can initiate. The following image illustrates how the DB2 engine works.

Display 1.1 Design of the DB2 Engine



How and When Connections Are Made

The interface to DB2 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 interface to DB2 determines when to make a connection to DB2 based on the type of open mode (read, update, or output mode) that a SAS application requests for the DB2 tables. The default behavior is as follows:

- the interface to DB2 shares the connection for all openings in read mode for each DB2 LIBNAME statement
- the interface to DB2 acquires a separate connection to DB2 for every opening in update or output mode.

You can change this default behavior by using the CONNECTION= option.

Several SAS applications require the interface to DB2 to query the DB2 system catalogs. When this type of query is required, the interface to DB2 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 50 for more information.

The Distributed Data Facility

Distributed Data Facility (DDF) is an optional product that enables z/OS DB2 applications to access data on other systems. The interface to DB2 supports both types of DDF: system-directed access (private protocol) and Distributed Relational Database Architecture.

System-directed access enables one z/OS DB2 subsystem to execute SQL statements on another z/OS DB2 subsystem. System-directed access uses a DB2-only private protocol. The interface to DB2 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, you must specify the LOCATION= option. When you specify this option, the three-level table name (*location.authid.table*) is used in the SQL statement that is generated by the interface to DB2. 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.

Distributed Relational Database Architecture (DRDA) is a set of protocols that enables a user to access distributed data. This enables the interface to DB2 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.

To connect to a DRDA remote server or location, the interface to DB2 uses an explicit connection. To establish an explicit connection, the interface to DB2 first connects to the local DB2 subsystem via an attachment facility (CAF or RRSAF). Then it issues an SQL CONNECT statement to connect from the local DB2 subsystem to the remote DRDA server prior to accessing data. To initiate a connection to a DRDA remote server, you must specify the connection option SERVER=. The SAS application uses a separate connection (specifying the SERVER= option) for each remote DRDA location.

DB2 Attachment Facilities (CAF and RRSAF)

By default, the SAS/ACCESS interface to DB2 uses the *Call Attachment Facility* (CAF) to make its connections to DB2. SAS supports multiple CAF connections for a SAS session. Thus, for a SAS server, all clients can have their own connections to DB2; multiple clients no longer have to share one connection. Because CAF does not support sign on, however, each connection that the SAS server makes to DB2 has the z/OS authorization identifier of the server, not the authorization identifier of the client for which the connection is made.

If you specify the DB2RRS system option, the interface to DB2 engine 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. SAS supports multiple RRSAF connections for a SAS session. RRSAF is a new feature in DB2 Version 5, Release 1, and the support for it by the interface to DB2 was new in Version 8 of SAS.

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 z/OS 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 or 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 is the user ID with which you are logged onto z/OS.

With RRSAF, the SAS server makes the connections for each client and the connections have the client's z/OS authorization identifier associated with them. This is only true for clients that were authenticated by the SAS server, which occurred when the client specified a user ID and password. Servers authenticate their clients when the clients provide their user IDs and passwords. Generally, this is the default way that servers are run. If a client connects to a SAS server without providing his user ID and password, then the identifier associated with his connections is that of the server (as with CAF) and not the identifier of the client.

Other than specifying DB2RRS at SAS start-up, there is nothing else that needs to be done in order to use RSSAF. The interface to DB2 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 many types of SAS procedures, the interface to DB2 must access the DB2 system catalogs for information. This information is limited to a list of all the tables for a specific authorization identifier. The interface generates the following SQL query in order to get information from the system catalogs:

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

Unless you specify the AUTHID= option, the AUTHID is the z/OS user ID that is associated with the job step.

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 fails.

Because querying the DB2 system catalogs can cause some locking contentions, the interface to DB2 initiates a separate connection for the query to the DB2 system catalogs. After the query has completed, a COMMIT WORK command is executed.