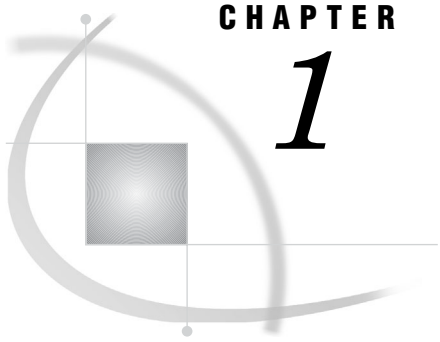

CHAPTER

1

Overview of SAS/ACCESS Interface to Relational Databases

About This Document

This document provides conceptual, reference, and usage information for the SAS/ACCESS interface to relational database management systems (DBMSs). The information in this document applies generally to all relational DBMSs that SAS/ACCESS software supports.

Because availability and behavior of SAS/ACCESS features vary from one interface to another, you should use the general information in this document with the DBMS-specific information in reference section of this document for your SAS/ACCESS interface.

This document is intended for applications programmers and end users who meet these conditions:

- familiar with the basics of their DBMS and its SQL (Structured Query Language)
- know how to use their operating environment
- can use basic SAS commands and statements

Database administrators might also want to read this document to understand how the interface is implemented and administered.

Methods for Accessing Relational Database Data

SAS/ACCESS Interface to Relational Databases is a family of interfaces—each licensed separately—with which you can interact with data in other vendor databases from within SAS. SAS/ACCESS provides these methods for accessing relational DBMS data.

- You can use the LIBNAME statement to assign SAS librefs to DBMS objects such as schemas and databases. After you associate a database with a libref, you can use a SAS two-level name to specify any table or view in the database. You can then work with the table or view as you would with a SAS data set.
- You can use the SQL pass-through facility to interact with a data source using its native SQL syntax without leaving your SAS session. SQL statements are passed directly to the data source for processing.
- You can use ACCESS and DBLOAD procedures for indirect access to DBMS data. Although SAS still supports these procedures for database systems and environments on which they were available for SAS 6, they are no longer the recommended method for accessing DBMS data.

See “Selecting a SAS/ACCESS Method” on page 4 for information about when to use each method.

Not all SAS/ACCESS interfaces support all of these features. To determine which features are available in your environment, see “Introduction” on page 77.

Selecting a SAS/ACCESS Method

Methods for Accessing DBMS Tables and Views

In SAS/ACCESS, you can often complete a task in several ways. For example, you can access DBMS tables and views by using the LIBNAME statement or the SQL pass-through facility. Before processing complex or data-intensive operations, you might want to test several methods first to determine the most efficient one for your particular task.

SAS/ACCESS LIBNAME Statement Advantages

You should use the SAS/ACCESS LIBNAME statement for the fastest and most direct method of accessing your DBMS data except when you need to use SQL that is not ANSI-standard. ANSI-standard SQL is required when you use the SAS/ACCESS library engine in the SQL procedure. However, the SQL pass-through facility accepts all SQL extensions that your DBMS provides.

Here are the advantages of using the SAS/ACCESS LIBNAME statement.

- Significantly fewer lines of SAS code are required to perform operations on your DBMS. For example, a single LIBNAME statement establishes a connection to your DBMS, lets you specify how data is processed, and lets you easily view your DBMS tables in SAS.
- You do not need to know the SQL language of your DBMS to access and manipulate data on your DBMS. You can use such SAS procedures as PROC SQL

or DATA step programming on any libref that references DBMS data. You can read, insert, update, delete, and append data. You can also create and drop DBMS tables by using SAS syntax.

- The LIBNAME statement gives you more control over DBMS operations such as locking, spooling, and data type conversion through the use of LIBNAME and data set options.
- The engine can optimize processing of joins and WHERE clauses by passing them directly to the DBMS, which takes advantage of the indexing and other processing capabilities of your DBMS. For more information, see "Overview of Optimizing Your SQL Usage" on page 43.
- The engine can pass some functions directly to the DBMS for processing.

SQL Pass-Through Facility Advantages

Here are the advantages of using the SQL pass-through facility.

- You can use SQL pass-through facility statements so the DBMS can optimize queries, particularly when you join tables. The DBMS optimizer can take advantage of indexes on DBMS columns to process a query more quickly and efficiently.
- SQL pass-through facility statements let the DBMS optimize queries when queries have summary functions (such as AVG and COUNT), GROUP BY clauses, or columns that expressions create (such as the COMPUTED function). The DBMS optimizer can use indexes on DBMS columns to process queries more rapidly.
- On some DBMSs, you can use SQL pass-through facility statements with SAS/AF applications to handle transaction processing of DBMS data. Using a SAS/AF application gives you complete control of COMMIT and ROLLBACK transactions. SQL pass-through facility statements give you better access to DBMS return codes.
- The SQL pass-through facility accepts all extensions to ANSI SQL that your DBMS provides.

SAS/ACCESS Features for Common Tasks

Here is a list of tasks and the features that you can use to accomplish them.

Table 1.1 SAS/ACCESS Features for Common Tasks

Task	SAS/ACCESS Features
Read DBMS tables or views	LIBNAME statement*
	SQL Pass-Through Facility
	View descriptors**
Create DBMS objects, such as tables	LIBNAME statement*
	DBLOAD procedure
	SQL Pass-Through Facility EXECUTE statement
Update, delete, or insert rows into DBMS tables	LIBNAME statement*
	View descriptors**
	SQL Pass-Through Facility EXECUTE statement

Task	SAS/ACCESS Features
Append data to DBMS tables	DBLOAD procedure with APPEND option
	LIBNAME statement and APPEND procedure*
	SQL Pass-Through Facility EXECUTE statement
	SQL Pass-Through Facility INSERT statement
List DBMS tables	LIBNAME statement and SAS Explorer window*
	LIBNAME statement and DATASETS procedure*
	LIBNAME statement and CONTENTS procedure*
	LIBNAME statement and SQL procedure dictionary tables*
Delete DBMS tables or views	LIBNAME statement and SQL procedure DROP TABLE statement*
	LIBNAME statement and DATASETS procedure DELETE statement*
	DBLOAD procedure with SQL DROP TABLE statement
	SQL Pass-Through Facility EXECUTE statement

* LIBNAME statement refers to the SAS/ACCESS LIBNAME statement.
** View descriptors refer to view descriptors that are created in the ACCESS procedure.

SAS Views of DBMS Data

SAS/ACCESS enables you to create a SAS view of data that exists in a relational database management system. A *SAS data view* defines a virtual data set that is named and stored for later use. A view contains no data, but rather describes data that is stored elsewhere. There are three types of SAS data views:

- *DATA step views* are stored, compiled DATA step programs.
- *SQL views* are stored query expressions that read data values from their underlying files, which can include SAS data files, SAS/ACCESS views, DATA step views, other SQL views, or relational database data.
- *SAS/ACCESS views* (also called view descriptors) describe data that is stored in DBMS tables. This is no longer a recommended method for accessing relational DBMS data. Use the CV2VIEW procedure to convert existing view descriptors into SQL views.

You can use all types of views as inputs into DATA steps and procedures. You can specify views in queries as if they were tables. A view derives its data from the tables or views that are listed in its FROM clause. The data accessed by a view is a subset or superset of the data in its underlying table(s) or view(s).

You can use SQL views and SAS/ACCESS views to update their underlying data if the view is based on only one DBMS table or if it is based on a DBMS view that is based on only one DBMS table and if the view has no calculated fields. You cannot use DATA step views to update the underlying data; you can use them only to read the data.

Your options for creating a SAS view of DBMS data are determined by the SAS/ACCESS feature that you are using to access the DBMS data. This table lists the recommended methods for creating SAS views.

Table 1.2 Creating SAS Views

Feature You Use to Access DBMS Data	SAS View Technology You Can Use
SAS/ACCESS LIBNAME statement	SQL view or DATA step view of the DBMS table
SQL Pass-Through Facility	SQL view with CONNECTION TO component

Choosing Your Degree of Numeric Precision

Factors That Can Cause Calculation Differences

Different factors affect numeric precision. This issue is common for many people, including SAS users. Though computers and software can help, you are limited in how precisely you can calculate, compare, and represent data. Therefore, only those people who generate and use data can determine the exact degree of precision that suits their enterprise needs.

As you decide the degree of precision that you want, you need to consider that these system factors can cause calculation differences:

- hardware limitations
- differences among operating systems
- different software or different versions of the same software
- different database management systems (DBMSs)

These factors can also cause differences:

- the use of finite number sets to represent infinite real numbers
- how numbers are stored, because storage sizes can vary

You also need to consider how conversions are performed—on, between, or across any of these system or calculation factors.

Examples of Problems That Result in Numeric Imprecision

Depending on the degree of precision that you want, calculating the value of r can result in a tiny residual in a floating-point unit. When you compare the value of **r** to 0.0, you might find that r≠0.0. The numbers are very close but not equal. This type of discrepancy in results can stem from problems in *representing*, *rounding*, *displaying*, and *selectively extracting* data.

Representing Data

Some numbers can be represented exactly, but others cannot. As shown in this example, the number 10.25, which terminates in binary, can be represented exactly.

```
data x;
     x=10.25;
      put x hex16.;
run;
```

The output from this DATA step is an exact number: 4024800000000000. However, the number 10.1 cannot be represented exactly, as this example shows.

```
data x;
      x=10.1;
      put x hex16.;
run;
```

The output from this DATA step is an inexact number: 4024333333333333.

Rounding Data

As this example shows, rounding errors can result from platform-specific differences. No solution exists for such situations.

```
data x;
      x=10.1;
      put x hex16.;
      y=100000;
      newx=(x+y)-y;
      put newx hex16.;
 run;
```

In Windows and Linux environments, the output from this DATA step is 4024333333333*3333* (8/10-byte hardware double). In the Solaris x64 environment, the output is 4024333333334*000* (8/8-byte hardware double).

Displaying Data

For certain numbers such as x.5, the precision of displayed data depends on whether you round up or down. Low-precision formatting (rounding down) can produce different results on different platforms. In this example, the same high-precision (rounding up) result occurs for X=8.3, X=8.5, or X=hex16. However, a different result occurs for X=8.1 because this number does not yield the same level of precision.

```
data;
      x=input('C047DFFFFFFFFFFF', hex16.);
      put x= 8.1 x= 8.3 x= 8.5 x= hex16.;
run;
```

Here is the output under Windows or Linux (high-precision formatting).

```
x=-47.8
x=-47.750 x=-47.7500
x=C047DFFFFFFFFFFF
```

Here is the output under Solaris x64 (low-precision formatting).

```
x=-47.7
 x=-47.750 x=-47.7500
x=C047DFFFFFFFFFFF
```

To fix the problem that this example illustrates, you must select a number that yields the next precision level—in this case, 8.2.

Selectively Extracting Data

Results can also vary when you access data that is stored on one system by using a client on a different system. This example illustrates running a DATA step from a Windows client to access SAS data in the z/OS environment.

```
data z(keep=x);
      x=5.2;
```

```
      output;
      y=1000;
      x=(x+y)-y;   /*almost 5.2 */
      output;
run;

proc print data=z;
run;
```

Here is the output this DATA step produces.

```
Obs     x
1      5.2
2      5.2
```

The next example illustrates the output that you receive when you execute the DATA step interactively under Windows or under z/OS.

```
data z1;
      set z(where=(x=5.2));
run;
```

Here is the corresponding z/OS output.

```
NOTE: There were 1 observations read from the data set WORK.Z.
WHERE x=5.2;
NOTE: The data set WORK.Z1 has 1 observations and 1 variables.
The DATA statement used 0.00 CPU seconds and 14476K.
```

In the above example, the expected count was not returned correctly under z/OS because the imperfection of the data and finite precision are not taken into account. You cannot use equality to obtain a correct count because it does not include the "almost 5.2" cases in that count. To obtain the correct results under z/OS, you must run this DATA step:

```
data z1;
      set z(where=(compfuzz(x,5.2,1e-10)=0));
run;
```

Here is the z/OS output from this DATA step.

```
NOTE: There were 2 observations read from the data set WORK.Z.
WHERE COMPFUZZ(x, 5.2, 1E-10)=0;
NOTE: The data set WORK.Z1 has 2 observations and 1 variables.
```

Your Options When Choosing the Degree of Precision That You Need

After you determine the degree of precision that your enterprise needs, you can refine your software. You can use macros, sensitivity analyses, or fuzzy comparisons such as extractions or filters to extract data from databases or from different versions of SAS.

If you are running SAS 9.2, use the COMPFUZZ (fuzzy comparison) function. Otherwise, use this macro.

```
/*****************************************************************************/
/* This macro defines an EQFUZZ operator.  The subsequent DATA step shows    */
/* how to use this operator to test for equality within a certain tolerance. */
/*****************************************************************************/
%macro eqfuzz(var1, var2, fuzz=1e-12);
abs((&var1 - &var2) / &var1) < &fuzz
```

```
%mend;

data _null_;
      x=0;
      y=1;
      do i=1 to 10;
         x+0.1;
      end;
      if x=y then put 'x exactly equal to y';
      else if %eqfuzz(x,y) then put 'x close to y';
      else put 'x nowhere close to y';
run;
```

When you read numbers in from an external DBMS that supports precision beyond 15 digits, you can lose that precision. You cannot do anything about this for existing databases. However, when you design new databases, you can set constraints to limit precision to about 15 digits or you can select a numeric DBMS data type to match the numeric SAS data type. For example, select the BINARY_DOUBLE type in Oracle (precise up to 15 digits) instead of the NUMBER type (precise up to 38 digits).

When you read numbers in from an external DBMS for noncomputational purposes, use the DBSASTYPE= data set option, as shown in this example.

```
libname ora oracle user=scott password=tiger path=path;
data sasdata;
      set ora.catalina2( dbsastype= ( c1='char(20)') ) ;
run;
```

This option retrieves numbers as character strings and preserves precision beyond 15 digits. For details, see the DBSASTYPE= data set option.

References

See these resources for more detail about numeric precision, including variables that can affect precision.

The Aggregate. 2008. "Numerical Precision, Accuracy, and Range." Aggregate.Org: *Unbridled Computing*. Lexington, KY: University of Kentucky. Available http://aggregate.org/NPAR.

IEEE. 2008. "IEEE 754: Standard for Binary Floating-Point Arithmetic." Available http://grouper.ieee.org/groups/754/index.html. This standard defines 32-bit and 64-bit floating-point representations and computational results.

SAS Institute Inc. 2007. TS-230. *Dealing with Numeric Representation Error in SAS Applications*. Cary, NC: SAS Institute Inc. Available http://support.sas.com/techsup/technote/ts230.html.

SAS Institute Inc. 2007. TS-654. *Numeric Precision 101*. Cary, NC: SAS Institute Inc. Available http://support.sas.com/techsup/technote/ts654.pdf. This document is an overview of numeric precision and how it is represented in SAS applications.