Introduction to the SAS/ACCESS Interface to Informix

This document includes details only about the SAS/ACCESS interface to Informix. It should be used as a supplement to the generic SAS/ACCESS documentation.
When you access Informix tables by using the SAS/ACCESS interface to Informix, the default Informix read isolation level is set for committed reads, and SAS spooling is on. Committed reads enable you to read rows unless another user or process is updating the rows. Reading in this manner does not lock the rows. SAS spooling guarantees that you get identical data each time you re-read a row because SAS buffers the rows after you read them the first time. This default environment is suitable for most users. If this default environment is unsuitable for your needs, see “Locking in the Informix Interface” on page 11.

To see the SQL statements that SAS issues to the Informix server, include the SASTRACE= option in your code:

\[
\text{option sastrace=',,,d';}
\]

If you use quotation marks in your Informix SQL statements, your DELIMIDENT= environment variable should be set to DELIMIDENT=YES, or your statements could be rejected by Informix. Because some of the SAS options that preserve case generate SQL statements that contain quotation marks, you should set DELIMIDENT=YES in your environment.

LIBNAME Statement Specifics for Informix

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

\[
\text{LIBNAME libref informix <connection-options> <LIBNAME-options>};
\]

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.

informix
- is the SAS/ACCESS engine name for the interface to Informix.

collection-options
- provide connection information and control how SAS manages the timing and concurrence of the connection to the DBMS. The connection options for the interface to Informix are:
  \[
  \text{USER= '<Informix-user-name>'}
  \]
  specifies the Informix user name that you use to connect to the database that contains the tables and views that you want to access. If you omit the USER= option, your operating environment account name is used, if applicable to your operating environment.
USING=<Informix-password'>
specifies the password that is associated with the Informix user. If you omit
the password, Informix uses the password in the /etc/password file.
USING= can also be specified with the PASSWORD= and PWD= aliases.

SERVER=<ODBC-data-source'>
specifies the ODBC data source to which you want to connect. An error
occurs if the SERVER= option is not set. For UNIX platforms, the data
source must be configured by modifying the .ODBC.ini file. See your ODBC
driver manual for details.

Note: For the SAS/ACCESS 9 interface to Informix, the Informix ODBC Driver
API is used to connect to Informix, and the connection options have been changed
accordingly. The DATABASE= option from SAS/ACCESS Version 8 has been
removed. If you need to specify a database, set it in the .ODBC.ini file. For
SERVER= options, instead of specifying the server name, as in Version 8, you
specify an ODBC data source name. Optionally, a user ID and password can be
used in conjunction with SERVER=.

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

Table 1.1 SAS/ACCESS LIBNAME Options for Informix

<table>
<thead>
<tr>
<th>Option</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACCESS=</td>
<td>none</td>
</tr>
<tr>
<td>CONNECTION=</td>
<td>SHAREDREAD</td>
</tr>
<tr>
<td>CONNECTION_GROUP=</td>
<td>none</td>
</tr>
<tr>
<td>DBCOMMIT=</td>
<td>1000 (insert) or 0 (update)</td>
</tr>
<tr>
<td>DBCONINIT=</td>
<td>none</td>
</tr>
<tr>
<td>DBCONTERM=</td>
<td>none</td>
</tr>
<tr>
<td>DBCREATE_TABLE_OPTS=</td>
<td>none</td>
</tr>
<tr>
<td>DBGEN_NAME=</td>
<td>DBMS</td>
</tr>
<tr>
<td>DBINDEX=</td>
<td>NO</td>
</tr>
<tr>
<td>DBLIBINIT=</td>
<td>none</td>
</tr>
<tr>
<td>DBLIBTERM=</td>
<td>none</td>
</tr>
<tr>
<td>DBNULLKEYS=</td>
<td>NO</td>
</tr>
<tr>
<td>DBPROMPT=</td>
<td>NO</td>
</tr>
<tr>
<td>DBSASLABEL=</td>
<td>COMPAT</td>
</tr>
<tr>
<td>DBSLICEPARAM=</td>
<td>THREADED_APPS,2 or 3</td>
</tr>
<tr>
<td>DEFER=</td>
<td>NO</td>
</tr>
<tr>
<td>DIRECT_EXE=</td>
<td>none</td>
</tr>
<tr>
<td>DIRECT_SQL=</td>
<td>YES</td>
</tr>
</tbody>
</table>
**Informix LIBNAME Statement Example**

In the following example, the libref MYDBLIB uses the Informix interface to connect to an Informix database:

```plaintext
libname mydblib informix user=testuser using=testpass server=testdsn;
```

In this example, USER=, USING=, and SERVER= are connection options.

---

**Data Set Options for Informix**

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

<table>
<thead>
<tr>
<th>Option</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>DBCOMMIT=</td>
<td>LIBNAME option setting</td>
</tr>
<tr>
<td>DBCONDITION=</td>
<td>none</td>
</tr>
<tr>
<td>DBCREATE_TABLE_OPTS=</td>
<td>LIBNAME option setting</td>
</tr>
<tr>
<td>DBFORCE=</td>
<td>NO</td>
</tr>
<tr>
<td>DBGEN_NAME=</td>
<td>DBMS</td>
</tr>
<tr>
<td>DBINDEX=</td>
<td>LIBNAME option setting</td>
</tr>
<tr>
<td>DBKEY=</td>
<td>none</td>
</tr>
<tr>
<td>DLBLABEL=</td>
<td>NO</td>
</tr>
<tr>
<td>DBMASTER=</td>
<td>none</td>
</tr>
<tr>
<td>DBNULL=</td>
<td><em>ALL</em>=_YES</td>
</tr>
</tbody>
</table>
### Pass-Through Facility Specifics for Informix

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 Informix are as follows:

- The `dbms-name` is `informix`.
- The `CONNECT` statement is optional when you are connecting to an Informix database if the `DBDATASRC` environment variable has been set. When you omit a `CONNECT` statement, an implicit connection is performed when the first `EXECUTE` statement or `CONNECTION TO` component is passed to the DBMS.
- You can connect to only one Informix database at a time. However, you can specify multiple `CONNECT` statements if they all connect to the same Informix database. If you use multiple connections, you must use an alias to identify the different connections. If you omit an alias, `informix` is automatically used.
- The `CONNECT` statement `database-connection-arguments` are identical to its `connection-options`.
- If you use quotation marks in your Informix Pass-Through statements, your `DELIMIDENT=` environment variable must be set to `DELIMIDENT=YES`, or your statements are rejected by Informix.

### Stored Procedures and the Pass-Through Facility

The Pass-Through Facility recognizes two types of stored procedures in Informix that perform only database functions. The methods for executing the two types of stored procedures are different.

<table>
<thead>
<tr>
<th>Option</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>DBNULLKEYS=</td>
<td>LIBNAME option setting</td>
</tr>
<tr>
<td>DBSASLABEL=</td>
<td>COMPAT</td>
</tr>
<tr>
<td>DBASTYPE=</td>
<td>see “Informix Data Types” on page 12</td>
</tr>
<tr>
<td>DBSLICE</td>
<td>none</td>
</tr>
<tr>
<td>DBSLICEPARM</td>
<td>THREADED_APPS,2 or 3</td>
</tr>
<tr>
<td>DBTYPE=</td>
<td>see “Informix Data Types” on page 12</td>
</tr>
<tr>
<td>ERLIMIT=</td>
<td>1</td>
</tr>
<tr>
<td>LOCKTABLE=</td>
<td>LIBNAME option setting</td>
</tr>
<tr>
<td>NULLCHAR=</td>
<td>SAS</td>
</tr>
<tr>
<td>NULLCHARVAL=</td>
<td>a blank character</td>
</tr>
<tr>
<td>PRESERVE_COL_NAMES=</td>
<td>LIBNAME option setting</td>
</tr>
<tr>
<td>SASDATEFMT=</td>
<td>DATETIME</td>
</tr>
<tr>
<td>SCHEMA=</td>
<td>LIBNAME option setting</td>
</tr>
</tbody>
</table>
Procedures that return no values to the calling application:

Stored procedures that do not return values can be executed directly by using the Informix SQL EXECUTE statement. Stored procedure execution is initiated with the Informix EXECUTE PROCEDURE statement. The following example executes the stored procedure make_table. The stored procedure has no input parameters and returns no values.

```
execute (execute procedure make_table())
by informix;
```

Procedures that return values to the calling application:

Stored procedures that return values must be executed by using the PROC SQL SELECT statement with a CONNECTION TO component. The following example executes the stored procedure read_address, which has one parameter, "Putnum".

The values that are returned by read_address serve as the contents of a virtual table for the PROC SQL SELECT statement.

```
select * from connection to informix
  (execute procedure read_address ("Putnum"));
```

For example, when you try to execute a stored procedure that returns values from a PROC SQL EXECUTE statement, you get the following error message:

```
execute (execute procedure read_address
  ("Putnum")) by informix;
```

ERROR: Informix EXECUTE Error: Procedure (read_address) returns too many values.

---

Command Restrictions for the Pass-Through Facility

Informix SQL contains extensions to the ANSI-89 standards. Some of these extensions, such as LOAD FROM and UNLOAD TO, are restricted from use by any applications other than the Informix DB-Access product. Specifying these extensions in the PROC SQL EXECUTE statement generates this error:

-201
A syntax error has occurred

---

Examples

The following example connects to Informix by using data source testdsn:

```
proc sql;
   connect to informix
      (user=SCOTT password=TIGER server=testdsn);
```

Note: You can use the DBDATASRC environment variable to specify the data source. △

The following example grants UPDATE and INSERT authority to user gomez on the Informix ORDERS table. Because the CONNECT statement is omitted, an implicit connection is made that uses a default value of informix as the connection alias and default values for the DATABASE= and SERVER= arguments. Informix is a case-sensitive database; therefore, the database object ORDERS is in uppercase, as it was created.
proc sql;
  execute (grant update, insert on ORDERS to gomez) by informix;
quit;

The following example connects to Informix and drops (that is, removes) the table TempData from the database. The alias Temp5 that is specified in the CONNECT statement is used in the EXECUTE statement’s BY clause.

proc sql;
  connect to informix as temp5
  (server=testdsn);
  execute (drop table tempdata) by temp5;
  disconnect from temp5;
quit;

The following example sends an SQL query, shown with highlighting, to the database for processing. The results from the SQL query serve as a virtual table for the PROC SQL FROM clause. In this example, DBCON is a connection alias.

proc sql;
  connect to informix as dbcon
    (user=testuser using=testpass
     server=testdsn);

  select *
    from connection to dbcon
    (select empid, lastname, firstname,
     hiredate, salary
     from employees
     where hiredate>='31JAN88');

  disconnect from dbcon;
quit;

The followings example gives the previous query a name and stores it as the PROC SQL view Samples.Hires88. The CREATE VIEW statement appears in highlighting.

libname samples 'SAS-data-library';

proc sql;
  connect to informix as mycon
    (user=testuser using=testpass
     server=testdsn);

  create view samples.hires88 as
    select *
      from connection to mycon
      (select empid, lastname, firstname,
       hiredate, salary from employees
       where hiredate>='31JAN88');

  disconnect from mycon;
quit;
The following example connects to Informix and executes the stored procedure `testproc`. The `select *` clause displays the results from the stored procedure.

```sql
proc sql;
  connect to informix as mydb
    (server=testdsn);
  select * from connection to mydb
    (execute procedure testproc('123456'));
  disconnect from mydb;
quit;
```

# Autopartitioning Scheme for Informix

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

## Overview

The autopartitioning method available for SAS/ACCESS to Informix is modeled after the MOD function method as described in the section about autopartitioning techniques in *SAS/ACCESS for Relational Databases: Reference*.

## Autopartitioning Restrictions

SAS/ACCESS to Informix places additional restrictions on which columns can be used for the partitioning column during the autopartitioning phase. Columns are partitioned as follows:

- INTEGER
- SMALLINT
- BIT
- TINYINT

DECIMALS with 0 scales columns may also be used as the partitioning column. Nullable columns are the least preferable.

## Using WHERE Clauses

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

```sas
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, and DBSLICEPARM= does not specify a maximum number of threads to use for the threaded read, the SAS/ACCESS interface to Informix defaults to three threads.

The following example demonstrates the use of DBSLICEPARM=, with the maximum number of threads set to five:

```sas
libname x informix user=dbitest using=dbigrp1 server=odbc15;
proc print data=x.dept(dbsliceparm=(ALL,5));
run;
```

Using DBSLICE=

You might achieve the best possible performance when using threaded reads by specifying an Informix specific DBSLICE= data set option in your SAS operation. The following example demonstrates the use of DBSLICE=:

```sas
libname x informix user=dbitest using=dbigrp1 server=odbc15;
data xottest;
set x.invoice(dbslice="amtbilled<10000000" "amtbilled>=10000000");
run;
```

Passing SAS Functions to Informix

The interface to Informix passes the following SAS functions to Informix for processing. See the section about optimizing SQL usage in SAS/ACCESS for Relational Databases: Reference for information.

- ABS
- ARCOS
- ARSIN
- ATAN
- ATAN2
- AVG
- COS
- DATE
- EXP
- INT
- LOG
- LOG10
- MAX
- MDY
- MIN
- SIN
Passing Joins to Informix

In order for a multiple libref join to pass to Informix, each of the following components of the LIBNAME statements must match exactly:

- user ID
- password
- server

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.

Temporary Table Support for Informix

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 establish the DBMS connection to support the creation and use of temporary tables, issue a LIBNAME statement with the connection options `CONNECTION_GROUP=connection-group` and `CONNECTION=GLOBAL`. This LIBNAME statement is required even if you connect to the database using the Pass-Through Facility CONNECT statement, because it establishes a connection group.

For every new PROC SQL step or LIBNAME statement, you must reissue a CONNECT statement with the `CONNECTION_GROUP=` option set to the same value so that the connection can be reused.

Terminating a Temporary Table

To terminate a temporary table, disassociate the libref by issuing the following statement:

```sas
libname libref clear;
```
Example

In the following Pass-Through example, joins are pushed to Informix:

```sas
libname x informix user=tester using=xxxxx server=dsn_name
    connection=global connection_group=mygroup;

proc sql;
    connect to informix (user=tester using=xxxxx server=dsn_name
                         connection=global connection_group=mygroup);
    execute (select * from t1 where (id >100)
             into scratch scr1 ) by informix;
    create table count2 as select * from connection to informix
        (select count(*) as totrec from scr1);
quit;

proc print data=count2;
run;

proc sql;
    connect to informix (user=tester using=xxxxx server=dsn_name
                         connection=global connection_group=mygroup);
    execute (select t2.fname, t2.lname, scr1.dept from t2, scr1 where
             (t2.id = scr1.id) into scratch t3 ) by informix;
quit;

libname x clear; /* connection closed, temp table closed */
```

Locking in the Informix Interface

In most situations, SAS spooling, which is on by default with the Informix interface, provides the data consistency you need.

The READ_ISOLATION_LEVEL= LIBNAME option enables you to control how the interface to Informix handles locks. This option can take the following values:

- **COMMITTED_READ**
  retrieves only committed rows. No locks are acquired, and rows can be locked exclusively for update by other users or processes. This is the default setting.

- **REPEATABLE_READ**
  gives you a shared lock on every row that is selected during the transaction. Other users or processes can also acquire a shared lock, but no other process can modify any row that is selected by your transaction. If you repeat the query during the transaction, you re-read the same information. The shared locks are released only when the transaction commits or rolls back. Another process cannot update or delete a row that is accessed by using a repeatable read.

- **DIRTY_READ**
  retrieves committed and uncommitted rows that might include phantom rows, which are rows that are created or modified by another user or process that might subsequently be rolled back. This type of read is most appropriate for tables that are not frequently updated.
CURSOR_STABILITY

gives you a shared lock on the selected row. Another user or process can acquire a
shared lock on the same row, but no process can acquire an exclusive lock to
modify data in the row. When you retrieve another row or close the cursor, the
shared lock is released.

If you set READ_ISOLATION_LEVEL= to REPEATABLE_READ or
CURSOR_STABILITY, it is recommended that you assign a separate libref and that you
clear that libref when you have finished working with the tables. This technique
minimizes the negative performance impact on other users that occurs when you lock
the tables. To clear the libref, include the following code:

    libname libref clear;

Note: For current Informix releases, READ_ISOLATION_LEVEL= is only valid
when transaction logging is enabled. If transaction logging is not enabled, an error is
generated when you use this option. Also, locks placed when
READ_ISOLATION_LEVEL= REPEATABLE_READ or CURSOR_STABILITY are not
freed until the libref is cleared.

To see the SQL locking statements that SAS issues to the Informix server, include
the SASTRACE= option in your code:

    option sastrace=’,,,d’;

For more details about Informix locking, see your Informix documentation.

---

**Naming Conventions for Informix**

The PRESERVE_COL_NAMES= and PRESERVE_TAB_NAMES= options determine
how the interface to Informix handles case sensitivity, spaces, and special characters.
See the section about the LIBNAME statement in *SAS/ACCESS for Relational
Databases: Reference* for information about these options.

The Informix naming conventions are as follows:

- Table and column names must begin with a letter or an underscore followed by
  letters, numbers, or underscores. However, if the name appears within quotation
  marks and PRESERVE_TAB_NAMES=YES (when applicable), it can start with
  any character.

- Table and column names can contain up to 32 characters for connecting to
  Informix Dynamic Server 200, and 18 characters for the other servers.

Note: Informix encourages users to utilize lower case for table and column names.
Several problems have been found in the Informix ODBC driver that result from using
upper case or mixed case.

Currently Informix has no schedule for fixing these known problems.

---

**Informix Data Types**

Every column in a table has a name and a data type. The data type tells Informix
how much physical storage to set aside for the column and the form in which the data
is stored.
Character Data

CHAR(n), NCHAR(n)
contains character string data from 1 to 32,767 characters in length and can
include tabs and spaces.

VARCHAR(m,n), NVARCHAR(m,n)
contains character string data from 1 to 255 characters in length.

TEXT
contains unlimited text data, depending on memory capacity.

BYTE
contains binary data of variable length.

Numeric Data

DECIMAL, MONEY, NUMERIC
contains numeric data with definable scale and precision. The amount of storage
that is allocated depends on the size of the number.

FLOAT, DOUBLE PRECISION
contains double-precision numeric data up to 8 bytes.

INTEGER
contains an integer up to 32 bits (from \(-2^{31}\) to \(2^{31}-1\)).

REAL, SMALLFLOAT
contains single-precision, floating-point numbers up to 4 bytes.

SERIAL
stores sequential integers up to 32 bits.

SMALLINT
contains integers up to 2 bytes.

INT8
contains an integer up to 64 bits (\(-2^{63-1}\) to \(2^{63-1}\)).

SERIAL8
contains sequential integers up to 64 bits.

Abstract Data

DATE
contains a calendar date in the form of a signed integer value.

DATETIME
contains a calendar date and time of day stored in 2 to 11 bytes, depending on
precision.

Note: When the DATETIME column is in an uncommon format (i.e.,
DATETIME MINUTE TO MINUTE or DATETIME SECOND TO SECOND), the
date and time values might not display correctly. △

INTERVAL
contains a span of time stored in 2 to 12 bytes, depending on precision.
Informix Null Values

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

If you do not indicate a default value for an Informix column, the default value is NULL. You can specify the keywords NOT NULL after the data type of the column when you create an Informix table to prevent NULL values from being stored in the column. When creating an Informix table with SAS/ACCESS, you can use the DBNULL= data set option to indicate whether NULL is a valid value for specified columns.

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

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

LIBNAME Statement Data Conversions

The following table shows the default SAS variable formats that SAS/ACCESS applies to Informix data types during input operations when you use the LIBNAME statement. You can override these default data types by using the DBTYPE= data set option on a specific data set.

Table 1.3 LIBNAME Statement: Default SAS Formats for Informix Data Types

<table>
<thead>
<tr>
<th>Informix Column Type</th>
<th>Default SAS Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHAR(n)</td>
<td>$n</td>
</tr>
<tr>
<td>DATE</td>
<td>DATE9</td>
</tr>
<tr>
<td>DATETIME**</td>
<td>DATETIME24.5</td>
</tr>
<tr>
<td>DECIMAL</td>
<td>m+2.n</td>
</tr>
<tr>
<td>DOUBLE PRECISION</td>
<td>none</td>
</tr>
<tr>
<td>FLOAT</td>
<td>none</td>
</tr>
<tr>
<td>INTEGER</td>
<td>none</td>
</tr>
<tr>
<td>INT8#</td>
<td>none</td>
</tr>
<tr>
<td>INTERVAL</td>
<td>$n</td>
</tr>
<tr>
<td>MONEY</td>
<td>none</td>
</tr>
<tr>
<td>NCHAR(n)</td>
<td>$n</td>
</tr>
<tr>
<td>NUMERIC</td>
<td>none</td>
</tr>
<tr>
<td>NVARCHAR(m,n)*</td>
<td>$m</td>
</tr>
<tr>
<td>REAL</td>
<td>none</td>
</tr>
<tr>
<td>SERIAL</td>
<td>none</td>
</tr>
<tr>
<td>SERIAL8#</td>
<td>none</td>
</tr>
<tr>
<td>SMALLFLOAT</td>
<td>none</td>
</tr>
<tr>
<td>Informix Column Type</td>
<td>Default SAS Format</td>
</tr>
<tr>
<td>----------------------</td>
<td>--------------------</td>
</tr>
<tr>
<td>SMALLINT</td>
<td>none</td>
</tr>
<tr>
<td>TEXT*</td>
<td>$n</td>
</tr>
<tr>
<td>VARCHAR($m,n)*</td>
<td>$m</td>
</tr>
</tbody>
</table>

* Only supported by Informix-Online databases

# The precision of a INT8 or SERIAL8 is 15 digit.

** If the Informix field qualifier specifies either HOUR, MINUTE, SECOND, or FRACTION as the largest unit, the value is converted to a SAS TIME value. All others, such as YEAR, MONTH, or DAY, are converted to a SAS DATETIME value.

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

Table 1.4 LIBNAME Statement: Default Informix Data Types for SAS Variable Formats

<table>
<thead>
<tr>
<th>SAS Variable Format</th>
<th>Informix Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>$w.</td>
<td>CHAR(w).</td>
</tr>
<tr>
<td>w. with SAS format name of NULL</td>
<td>DOUBLE</td>
</tr>
<tr>
<td>w.d with SAS format name of NULL</td>
<td>DOUBLE</td>
</tr>
<tr>
<td>all other numerics</td>
<td>DOUBLE</td>
</tr>
<tr>
<td>datetime w.d</td>
<td>DATETIME YEAR TO FRACTION(5)</td>
</tr>
<tr>
<td>date w.</td>
<td>DATE</td>
</tr>
<tr>
<td>time.</td>
<td>DATETIME HOUR TO SECOND</td>
</tr>
</tbody>
</table>

Pass-Through Facility Data Conversions

The Pass-Through Facility uses the same default conversion formats as the LIBNAME statement. See “LIBNAME Statement Data Conversions” on page 14 for the conversion tables.

Overview of Informix Servers

Informix Database Servers

There are two types of Informix database servers, the Informix-Online and Informix-SE servers. Informix-Online database servers can support many users and provide tools that ensure high availability, high reliability, and that support critical applications. Informix-SE database servers are designed to manage relatively small databases that are used privately by individuals or shared among a small number of users.
Using the DBDATASRC Environment Variables

The Pass-Through Facility supports the environment variable DBDATASRC, which is an extension to the Informix environment variable. If you set DBDATASRC, you can omit the CONNECT statement. The value of DBDATASRC is used instead of the SERVER= argument in the CONNECT statement. The syntax for setting DBDATASRC is like the syntax of the SERVER= argument:

Bourne shell:
```bash
DBDATABASE='testdsn' export DBDATASRC
```

C shell:
```bash
setenv DBDATASRC testdsn
```

If you set DBDATASRC, you can issue a PROC SQL SELECT or EXECUTE statement without first connecting to Informix with the CONNECT statement.

If you omit the CONNECT statement, an implicit connection is performed when the SELECT or EXECUTE statement is passed to Informix.

If you create an SQL view without an explicit CONNECT statement, the view can dynamically connect to different databases, depending on the value of the DBDATASRC environment variable.

Using Fully Qualified Table Names

Informix supports a connection to only one database. If you have data that spans multiple databases, you must use fully qualified table names to work within the Informix single-connection constraints.

In the following example, the tables Tab1 and Tab2 reside in different databases, MyDB1 and MyDB2, respectively.

```sas
proc sql;
  connect to informix
  (database=corpdb server=online);

  create view tablv as
  select * from connection
  to informix
  (select * from mydb1.tab1);

  create view tab2v as
  select * from connection
  to informix
  (select * from mydb2.tab2);

quit;

data getboth;
  merge tablv tab2v;
  by common;
run;
```

Because the tables reside in separate databases, you cannot connect to each database with a PROC SQL CONNECT statement and then retrieve the data in a single step. Using the fully qualified table name (that is, `database.table`) enables you to use any Informix database in the CONNECT statement and access Informix tables in the same or different databases in a single SAS procedure or DATA step.