CHAPTER
1

SAS/ACCESS for Oracle

Introduction to the SAS/ACCESS Interface to Oracle

This document includes details *only* about the SAS/ACCESS Interface to Oracle. It should be used as a supplement to the main SAS/ACCESS documentation, *SAS/ACCESS for Relational Databases: Reference*.

LIBNAME Statement Specifics for Oracle

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

LIBNAME *libref* **oracle** <*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.

oracle
is the SAS/ACCESS engine name for the interface to Oracle.

connection-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 Oracle are as follows:

USER=<'>*Oracle-user-name*<'>
specifies an optional Oracle user name. If the user name contains blanks or national characters, enclose it in quotation marks. If you omit an Oracle user name and password, the default Oracle user ID OPS$*sysid* is used, if it is enabled. USER= must be used with PASSWORD=.

PASSWORD=<'>*Oracle-password*<'>
specifies an optional Oracle password that is associated with the Oracle user name. If you omit PASSWORD=, the password for the default Oracle user ID OPS$sysid is used, if it is enabled. PASSWORD= must be used with USER=.

PATH=<'>*Oracle-database-specification*<'>
specifies the Oracle driver, node, and database. Aliases are required if you are using SQL*Net Version 2.0 or later. In some operating environments, you can enter the information that is required by the PATH= statement before invoking SAS.

SAS/ACCESS uses the same Oracle path designation that you use to connect to Oracle directly. See your database administrator to determine the databases that have been set up in your operating environment, and to determine the default values if you do not specify a database. On UNIX systems, the TWO_TASK environment variable is used, if set. If neither the PATH= nor the TWO_TASK values have been set, the default value is the local driver.

If you specify the appropriate system options or environment variables for Oracle, you can often omit the connection options from your LIBNAME statements. See your Oracle documentation for details.

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 LIBNAME options that are supported for Oracle, 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 for Oracle

Option	Default Value
ACCESS=	none
CONNECTION=	SHAREDREAD
CONNECTION_GROUP=	none
DBCOMMIT=	1000 when inserting rows; 0 when updating rows, deleting rows, or appending rows to an existing table
DBCONINIT=	none
DBCONTERM=	none
DBCREATE_TABLE_OPTS=	none
DBGEN_NAME=	DBMS
DBINDEX=	NO Use this option only when the object is a TABLE, not a VIEW. Use DBKEY when you do not know whether the object is a TABLE.
DBLIBINIT=	none
DBLIBTERM=	none
DBLINK=	the local database
DBMAX_TEXT=	1024
DBNULLKEYS=	YES
DBPROMPT=	NO
DBSLICEPARM=	THREADED_APPS,2
DEFER=	NO
DIRECT_EXE=	none
DIRECT_SQL=	YES
INSERTBUFF=	10
LOCKWAIT=	YES
MULTI_DATASRC_OPT=	NONE
ORACLE_UPD_NOWHERE=	YES
PRESERVE_COL_NAMES=	NO
PRESERVE_TAB_NAMES=	NO
READBUFF=	250
READ_ISOLATION_LEVEL=	see “Locking in the Oracle Interface” on page 19

Option	Default Value
READ_LOCK_TYPE=	NOLOCK
REREAD_EXPOSURE=	NO
SCHEMA=	SAS accesses objects in the default and public schemas
SHOW_SYNONYMS=	NO
SPOOL=	YES
UPDATE_ISOLATION_LEVEL=	see “Locking in the Oracle Interface” on page 19
UPDATE_LOCK_TYPE=	NOLOCK
UPDATEBUFF=	1
UTILCONN_TRANSIENT=	NO

Oracle LIBNAME Statement Examples

In the following example, the connection is made using default settings for the connection options. If you specify the appropriate system options or environment variables for Oracle, you can often omit the connection options from your LIBNAME statements. See your Oracle documentation for details.

```
libname myoralib oracle;
```

In the following example, the libref MYDBLIB uses the SAS/ACCESS interface to Oracle to connect to an Oracle database. The SAS/ACCESS connection options are USER=, PASSWORD=, and PATH=. PATH= specifies an alias for the database specification (as required by SQL*Net).

```
libname mydblib oracle user=testuser password=testpass path=hrdept_002;

proc print data=mydblib.employees;
   where dept='CSR010';
run;
```

Data Set Options for Oracle

The following table describes all of the data set options that are supported for the Oracle interface. Default values are provided where applicable. See the section about data set options in *SAS/ACCESS for Relational Databases: Reference* for general information about these options.

Table 1.2 Data Set Options for Oracle

Option	Default Value
BL_BADFILE=	creates a file in the current directory or with the default file specifications
BL_CONTROL=	creates a file in the current directory or with the default file specifications

Option	Default Value
BL_DATAFILE=	creates a file in the current directory or with the default file specifications
BL_DELETE_DATAFILE=	YES
BL_DIRECT_PATH=	YES
BL_DISCARDFILE=	creates a file in the current directory or with the default file specifications
BL_INDEX_OPTIONS=	the current SQL*Loader Index options with bulk-loading
BL_LOAD_METHOD=	When loading an empty table, the default value is INSERT; when loading a table that contains data, the default value is APPEND.
BL_LOG=	If there is no pre-existing log file, the default action is to create a log file in the current directory or the default file specifications. If there is already a log file, the Oracle bulk loader reuses the file, replacing the contents with information from the new load.
BL_OPTIONS=	ERRORS=1000000
BL_PRESERVE_BLANKS=	NO
BL_RECOVERABLE=	YES
BL_SQLLDR_PATH=	sqldr
BL_SUPPRESS_NULLIF=	NO
BULKLOAD=	NO
DBCOMMIT=	the current LIBNAME option setting
DBCONDITION=	none
DBCREATE_TABLE_OPTS=	the current LIBNAME option setting
DBFORCE=	NO
DBGEN_NAME=	DBMS
DBINDEX=	the current LIBNAME option setting
DBKEY=	none
DBLABEL=	NO
DBLINK=	the current LIBNAME option setting
DBMASTER=	none
DBMAX_TEXT=	1024
DBNULL=	YES
DBNULLKEYS=	the current LIBNAME option setting
DBPROMPT=	the current LIBNAME option setting
DBSASTYPE=	see "Data Types for Oracle Servers" on page 20
DBSLICE=	none
DBSLICEPARM=	THREADED_APPS,2

Option	Default Value
DBTYPE=	see "LIBNAME Statement Data Conversions" on page 22
ERRLIMIT=	1
INSERTBUFF=	the current LIBNAME option setting
NULLCHAR=	SAS
NULLCHARVAL=	a blank character
OR_PARTITION=	an Oracle table partition name
OR_UPD_NOWHERE=	the current LIBNAME option setting
ORHINTS=	no hints
PRESERVE_COL_NAMES=	current LIBNAME option setting
READ_ISOLATION_LEVEL=	the current LIBNAME option setting
READ_LOCK_TYPE=	the current LIBNAME option setting
READBUFF=	the current LIBNAME option setting
SASDATEFORMAT=	DATETIME20.0
SCHEMA=	the current LIBNAME option setting
UPDATE_ISOLATION_LEVEL=	the current LIBNAME option setting
UPDATE_LOCK_TYPE=	the current LIBNAME option setting
UPDATEBUFF=	the current LIBNAME option setting

Pass-Through Facility Specifics for Oracle

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 Oracle are as follows:

- The *dbms-name* is **`oracle`**.
- The CONNECT statement is optional. If you omit the CONNECT statement, an implicit connection is made with your OPS$*sysid*, if it is enabled. When you omit a CONNECT statement, an implicit connection is performed when the first EXECUTE statement or CONNECTION TO component is passed to Oracle. In this case you must use the default DBMS name **`oracle`**.
- The interface to Oracle can connect to multiple databases (both local and remote) and to multiple user IDs. If you use multiple simultaneous connections, you must use an *alias* argument to identify each connection. If you do not specify an alias, the default alias, **`oracle`**, is used.
- The *database-connection-arguments* for the CONNECT statement are as follows:

  USER=<'>*Oracle-user-name*<'>
  : specifies an optional Oracle user name. If you specify USER=, you must also specify PASSWORD=.

  PASSWORD= <'>*Oracle-password*<'>
  : specifies an optional Oracle password that is associated with the Oracle user name. If you omit an Oracle password, the default Oracle user ID OPS$*sysid* is used, if it is enabled. If you specify PASSWORD=, you must also specify USER=.

ORAPW= is an alias for this option.

Note: If you do not wish to enter your Oracle password in uncoded text, see PROC PWENCODE for a method to encode it. △

BUFFSIZE=*number-of-rows*
: specifies the number of rows to retrieve from an Oracle table or view with each fetch. Using this argument can improve the performance of any query to Oracle.

By setting the value of the BUFFSIZE= argument in your SAS programs, you can find the optimal number of rows for a given query on a given table. The default buffer size is 250 rows per fetch. The limit is 32,767 rows per fetch, although a practical limit for most applications is less, depending on the available memory.

PRESERVE_COMMENTS
: enables you to pass additional information (called *hints*) to Oracle for processing. These hints might direct the Oracle query optimizer to choose the best processing method based on your hint.

You specify PRESERVE_COMMENTS as an argument in the CONNECT statement. Then you specify the hints in the CONNECTION TO component's Oracle SQL query. The hints are entered as comments in the SQL query and are passed to and processed by Oracle.

PATH=<'>*Oracle-database-specification*<'>
: specifies the Oracle driver, node, and database. Aliases are required if you are using SQL*Net Version 2.0 or later. In some operating environments, you can enter the information that is required by the PATH= statement before invoking SAS.

SAS/ACCESS uses the same Oracle path designation that you use to connect to Oracle directly. See your database administrator to determine the path designations that have been set up in your operating environment, and to determine the default value if you do not specify a path designation. On UNIX systems, the TWO_TASK environment variable is used, if set. If neither PATH= nor TWO_TASK have been set, the default value is the local driver.

Examples

The following example uses the alias DBCON for the DBMS connection (the connection alias is optional):

```
proc sql;
   connect to oracle as dbcon
        (user=testuser password=testpass buffsize=100
         path='myorapath');
quit;
```

The following example connects to Oracle and sends it two EXECUTE statements to process.

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

The following example performs a query, shown in highlighted text, on the Oracle table CUSTOMERS:

```
proc sql;
connect to oracle (user=testuser password=testpass);
select *
   from connection to oracle
     (select * from customers
      where customer like '1%');
    disconnect from oracle;
quit;
```

In this example, the PRESERVE_COMMENTS argument is specified after the USER= and PASSWORD= arguments. The Oracle SQL query is enclosed in the required parentheses. The SQL INDX command identifies the index for the Oracle query optimizer to use in processing the query. Note that multiple hints are separated with blanks.

```
proc sql;
connect to oracle as mycon(user=testuser
        password=testpass preserve_comments);
select *
   from connection to mycon
     (select /* +indx(empid) all_rows */
          count(*) from employees);
quit;
```

Autopartitioning Scheme for Oracle

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

Note: Threaded reads for the Oracle engine are not supported under MVS (z/OS). △

Overview

In the absence of user specified partitioning from the DBSLICE= option, the SAS/ACCESS interface to Oracle attempts to use its own partitioning techniques. The partitioning technique it chooses depends on whether the table is physically partitioned on the Oracle server.

Partitioned Oracle Tables

If you are dealing with a partitioned Oracle table, it is recommended that you allow the Oracle engine to partition the table for you. The Oracle engine will gather all of the partition information needed to do a threaded read on the table.

A partitioned Oracle table is a good candidate for a threaded read, because each of the partitions in the table can be read in parallel without much contention for disk resources. If the Oracle engine determines that the table is partitioned, it makes the same number of connections to the server as there are partitions. Each connection retrieves rows from a single partition.

For example, assume a SALES table was created in Oracle as follows:

```
CREATE TABLE SALES (acct_no NUMBER(5),
 acct_name CHAR(30), amount_of_sale NUMBER(6), qtr_no INTEGER)
PARTITION BY RANGE (qtr_no)
(PARTITION sales1 VALUES LESS THAN (2) TABLESPACE ts0,
PARTITION sales2 VALUES LESS THAN (2) TABLESPACE ts1,
PARTITION sales3 VALUES LESS THAN (2) TABLESPACE ts2,
PARTITION sales4 VALUES LESS THAN (2) TABLESPACE ts3)
```

Performing a threaded read on this table causes SAS to make four separate connections to the Oracle server. Each connection reads from each of the partitions. Turning SASTRACE on shows you the SQL that is generated for each connection:

```
libname x oracle user=testuser path=oraserver;
data new;
set x.SALES (DBSLICEPARM=ALL);
run;

ORACLE: SELECT "ACCT_NO","ACCT_NAME", "AMOUNT_OF_SALE", "QTR_NO" FROM SALES
partition (SALES2)
ORACLE: SELECT "ACCT_NO","ACCT_NAME", "AMOUNT_OF_SALE", "QTR_NO" FROM SALES
partition (SALES3)
ORACLE: SELECT "ACCT_NO","ACCT_NAME", "AMOUNT_OF_SALE", "QTR_NO" FROM SALES
partition (SALES1)
ORACLE: SELECT "ACCT_NO","ACCT_NAME", "AMOUNT_OF_SALE", "QTR_NO" FROM SALES
partition (SALES4)
```

The number of threads used to read the table in parallel is determined by the second parameter of the DBSLICEPARM= option. The number of connections made to the Oracle server for retrieving rows from the table is determined by the number of partitions on the table.

Non-partitioned Oracle Tables

If the table is not partitioned, and the DBSLICE= option is not specified, Oracle resorts to the MOD function discussed in the section on threaded reads in *SAS/ACCESS for Relational Databases: Reference*. With this technique, the engine makes *N* connections, and each connection retrieves rows based on a WHERE clause as follows:

```
WHERE ABS(MOD(ModColumn,N))=R
```

- ModColumn is a column in the table which is of type integer and is not used in any user specified WHERE clauses. (The engine selects this column. If you do not think this is the ideal partitioning column, you can use DBSLICE= to override this default behavior.)
- R varies from 0 to (*N*-1) for each of the *N* WHERE clauses.
- *N* defaults to 2, and *N* can be overridden with the second parameter in DBSLICEPARM=.

The Oracle engine selects the ModColumn to use in this technique. Any numeric column with zero scale value can qualify as the ModColumn. However, if a primary key column is present, it is preferred over all others. Generally, values in the primary key column are in a serial order and will yield an equal number of rows for each connection.

An example illustrates this point:

```
create table employee (empno number(10) primary key,
  empname varchar2(20), hiredate date,
  salary number(8,2), gender char(1));
```

Performing a threaded read on this table causes Oracle to make two separate connections to the Oracle server. SAS tracing shows the SQL generated for each connection:

```
data new;
set x.EMPLOYEE(DBSLICPARM=ALL);
run;
ORACLE: SELECT "EMPNO", "EMPNAME", "HIREDATE", "SALARY", "GENDER"
FROM EMPLOYEE WHERE ABS(MOD("EMPNO",2))=0
ORACLE: SELECT "EMPNO", "EMPNAME", "HIREDATE", "SALARY", "GENDER"
FROM EMPLOYEE WHERE ABS(MOD("EMPNO",2))=1
```

EMPNO, the primary key, is selected as the MOD column.

The success of MOD depends on the distribution of the values within the selected ModColumn and the value of *N*. Ideally, the rows will be distributed evenly among the threads.

Performance Summary

There are times you might not see an improvement in performance with the MOD technique. It is possible that the engine might not be able to find a column that qualifies as a good MOD column. In these situations, you can explicitly specify DBSLICE= to force a threaded read and improve performance.

It is a good policy to let the engine autopartition and intervene with DBSLICE= only when necessary.

Temporary Table Support for Oracle

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

A temporary table in Oracle persists just like a regular table, but contains either session specific or transaction specific data. Whether the data is session or transaction specific is determined by what is specified with the ON COMMIT keyword when you create the temporary table.

In the SAS context, you must use the LIBNAME option CONNECTION=SHARED in order for the data in a temporary table to persist over procedure and DATA step boundaries. Without this option, the temporary table will persist but the data within it will not.

If you have a SAS data set and you want to join it with an Oracle table to generate a report, the join normally occurs in SAS. However, using a temporary table you can also have the join occur on the Oracle server.

Syntax

The syntax to create a temporary table whose data is transaction specific (default) is as follows:

CREATE GLOBAL TEMPORARY TABLE *table name* **ON COMMIT DELETE ROWS**

The syntax to create a temporary table whose data is session specific is as follows:

CREATE GLOBAL TEMPORARY TABLE *table name* **ON COMMIT PRESERVE ROWS**

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

In the following example, a temporary table, TEMPTRANS, is created in Oracle to match the TRANS SAS data set (using the Pass-Through Facility):

```
proc sql;
   connect to oracle (user=scott pw=tiger path=oraclev9);
   execute (create global temporary table TEMPTRANS
           (empid number, salary number)) by oracle;
quit;

libname ora oracle user=scott pw=tiger path=oracle9 connection=shared;

/* load the data from the TRANS table into the Oracle temporary table */
proc append base=ora.TEMPTRANS set TRANS;
run;

proc sql;
/* do the join on the DBMS server */
   select lastname, firstname, salary from ora.EMPLOYEES T1, ora.TEMPTRANS  T2
          where  T1.empno=T2.empno;
quit;
```

ACCESS Procedure Specifics for Oracle

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

The Oracle interface supports all of the ACCESS procedure statements. The ACCESS procedure specifics for Oracle are as follows:

- The PROC ACCESS step DBMS= value is **`Oracle`**.
- The *database-description-statements* used by PROC ACCESS are as follows:

  USER=<'>*Oracle-user-name*<'>
  : specifies an optional Oracle user name. If you omit an Oracle password and user name, the default Oracle user ID OPS$*sysid* is used if it is enabled. If you specify USER=, you must also specify ORAPW=.

  ORAPW= <'>*Oracle-password*<'>
  : specifies an optional Oracle password that is associated with the Oracle user name. If you omit ORAPW=, the password for the default Oracle user ID OPS$sysid is used, if it is enabled. If you specify ORAPW=, you must also specify USER=.

  PATH=<'>*Oracle-database-specification*<'>
  : specifies the Oracle driver, node, and database. Aliases are required if you are using SQL*Net Version 2.0 or later. In some operating environments, you can enter the information that is required by the PATH= statement before invoking SAS.

    SAS/ACCESS uses the same Oracle path designation that you use to connect to Oracle directly. See your database administrator to determine the path designations that have been set up in your operating environment, and to determine the default value if you do not specify a path designation. On UNIX systems, the TWO_TASK environment variable is used, if set. If neither PATH= nor TWO_TASK have been set, the default value is the local driver.

- The PROC ACCESS step TABLE= statement is as follows:

  TABLE= <'><*Oracle-table-name*><'>;
  : specifies the name of the Oracle table or Oracle view on which the access descriptor is based. This statement is required. The *Oracle-table-name* argument can be up to 30 characters long and must be a valid Oracle table name. If the table name contains blanks or national characters, enclose it in quotation marks.

Examples

The following example creates an access descriptor and a view descriptor based on Oracle data.

```
options linesize=80;

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

proc access dbms=oracle;

/* create access descriptor */

   create adlib.customer.access;
   user=testuser;
   orapw=testpass;
   table=customers;
   path='myorapath';
   assign=yes;
   rename customer=custnum;
   format firstorder date9.;
   list all;

/* create view descriptor */

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

The following example creates another view descriptor that is based on the ADLIB.CUSTOMER access descriptor. The view is then printed.

```
/* create socust view */

proc access dbms=oracle accdesc=adlib.customer;
   create vlib.socust.view;
   select customer state name contact;
   subset where state in ('NC', 'VA', 'TX');
run;

/* print socust view */

proc print data=vlib.socust;
title 'Customers in Southern States';
run;
```

DBLOAD Procedure Specifics for Oracle

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

The Oracle interface supports all of the DBLOAD procedure statements. The DBLOAD procedure specifics for Oracle are as follows:

- The PROC DBLOAD step DBMS= value is **`Oracle`**.
- The *database-description-statements* used by PROC DBLOAD are as follows:

USER=<'>*Oracle-user-name*<'>
: specifies an optional Oracle user name. If you omit an Oracle password and user name, the default Oracle user ID OPS$*sysid* is used if it is enabled. If you specify USER=, you must also specify ORAPW=.

ORAPW= <'>*Oracle-password*<'>
: specifies an optional Oracle password that is associated with the Oracle user name. If you omit ORAPW=, the password for the default Oracle user ID OPS$sysid is used, if it is enabled. If you specify ORAPW=, you must also specify USER=.

PATH=<'>*Oracle-database-specification*<'>
: specifies the Oracle driver, node, and database. Aliases are required if you are using SQL*Net Version 2.0 or later. In some operating environments, you can enter the information that is required by the PATH= statement before invoking SAS.

SAS/ACCESS uses the same Oracle path designation that you use to connect to Oracle directly. See your database administrator to determine the path designations that have been set up in your operating environment, and to determine the default value if you do not specify a path designation. On UNIX systems, the TWO_TASK environment variable is used, if set. If neither PATH= nor TWO_TASK have been set, the default value is the local driver.

TABLESPACE= <'>*Oracle-tablespace-name*<'>;
specifies the name of the Oracle tablespace where you want to store the new table. The *Oracle-tablespace-name* argument can be up to 18 characters long and must be a valid Oracle tablespace name. If the name contains blanks or national characters, enclose the entire name in quotation marks.

If TABLESPACE= is omitted, the table is created in your default tablespace that is defined by the Oracle database administrator at your site.

□ The PROC DBLOAD step TABLE= statement is as follows:

TABLE= <'><*Oracle-table-name*><'>;
specifies the name of the Oracle table or Oracle view on which the access descriptor is based. This statement is required. The *Oracle-table-name* argument can be up to 30 characters long and must be a valid Oracle table name. If the table name contains blanks or national characters, enclose the name in quotation marks.

Examples

The following example creates a new Oracle table, EXCHANGE, from the DLIB.RATEOFEX data file. An access descriptor, ADLIB.EXCHANGE, based on the new table, is also created. The PATH= statement uses an alias to connect to a remote Oracle7 Server database.

The SQL statement in the second DBLOAD procedure sends an SQL GRANT statement to Oracle. You must be granted Oracle privileges to create new Oracle tables or to grant privileges to other users. The SQL statement is in a separate procedure because you cannot create a DBMS table and reference it within the same DBLOAD step. The new table is not created until the RUN statement is processed at the end of the first DBLOAD step.

Note: The DLIB.RATEOFEX data set is included in the sample data shipped with your software. △

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

proc dbload dbms=oracle data=dlib.rateofex;
   user=testuser;
   orapw=testpass;
   path='myorapath';
   table=exchange;
   accdesc=adlib.exchange;
   rename fgnindol=fgnindolar 4=dolrsinfgn;
   nulls updated=n fgnindol=n 4=n country=n;
   load;
run;

proc dbload dbms=oracle;
```

```
   user=testuser;
   orapw=testpass;
   path='myorapath';
   sql grant select on testuser.exchange to pham;
run;
```

The next example uses the APPEND option to append rows from the INVDATA data set to an existing Oracle table named INVOICE.

```
proc dbload dbms=oracle data=invdata append;
  user=testuser;
  orapw=testpass;
  path='myorapath';
  table=invoice;
  load;
run;
```

Note: This example uses a previously created data set, INVDATA. △

Maximizing Oracle Performance

There are several measures you can take to optimize performance when using the SAS/ACCESS interface to Oracle. See the section about performance considerations in *SAS/ACCESS for Relational Databases: Reference* for general information about improving performance when using SAS/ACCESS engines.

The SAS/ACCESS interface to Oracle has several options that you can use to further improve performance. See the INSERTBUFF=, UPDATEBUFF= and READBUFF= LIBNAME options for tips on multi-row processing. See "Oracle Bulk Loading" on page 17 for instructions on using Oracle's SQL*Loader to increase performance when loading rows of data into Oracle tables.

Note: If you choose the transactional inserting of rows (specify BULKLOAD=NO), you can improve performance by inserting multiple rows at a time. This performance enhancement is comparable to using the Oracle SQL*Loader Conventional Path Load. For more information about inserting multiple rows, see the INSERTBUFF= option. △

Passing SAS Functions to Oracle

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

ABS

ARCOS (ACOS)

ARSIN (ASIN)

ATAN

CEIL

COS

COSH

DATETIME (SYSDATE)

EXP

FLOOR

LOG

LOG10 (LOG)

LOG2 (LOG)

LOWCASE (LOWER)

SIGN

SIN

SINH

SOUNDEX

SQRT

STRIP (TRIM)

TAN

TANH

TRIMN (RTRIM)

TRANSLATE

UPCASE (UPPER)

SUM

COUNT

AVE

MIN

MAX

Passing Joins to Oracle

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

user ID

password

path

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.

Oracle Bulk Loading

The SAS/ACCESS interface to Oracle can call the Oracle SQL*Loader (SQLLDR) when you set the data set option BULKLOAD=YES. The support for Oracle's bulk loader provides superior load performance, enabling you to rapidly move data from a SAS file into an Oracle table. In future releases, SAS/ACCESS software will continue to make use of powerful Oracle tools to improve its loading performance.

The bulk loading data set options for Oracle are listed below. See the section about data set options in *SAS/ACCESS for Relational Databases: Reference* for additional information about these options. Note that they all begin with BL_ (for BULKLOAD):

BL_BADFILE=

BL_CONTROL=

BL_DATAFILE=

BL_DELETE_DATAFILE=

BL_DIRECT_PATH=

BL_DISCARDFILE=

BL_INDEX_OPTIONS=

BL_LOAD_METHOD=

BL_LOG=

BL_OPTIONS=

BL_PARFILE=

BL_PRESERVE_BLANKS=

BL_RECOVERABLE=

BL_SQLLDR_PATH=

BL_SUPPRESS_NULLIF=

BULKLOAD=

BULKLOAD= invokes Oracle's bulk loader, enabling the Oracle engine to move data from a SAS file into an Oracle table using SQL*Loader (SQLLDR).

Note: SQL*Loader Direct path load has a number of limitations. Please refer to the Oracle utilities manual for details.

When using bulk load, you should consult the SQL*Loader log file (rather than the SAS log) for information about the load. △

Oracle Bulk Loading: Interactions with Other Options

When BULKLOAD=YES, the following is true:

- The DBCOMMIT=, DBFORCE=, ERRLIMIT=, and INSERTBUFF= options are ignored.
- If NULLCHAR=SAS, and the NULLCHARVAL value is blank, then the SQL*Loader attempts to insert a NULL instead of a NULLCHARVAL value.
- If NULLCHAR=NO, and the NULLCHARVAL value is blank, then the SQL*Loader attempts to insert a NULL even if the DBMS does not allow NULL.

  To avoid this result, set BL_PRESERVE_BLANKS=YES or set NULLCHARVAL to a non-blank value (and then replace the non-blank value with blanks after processing, if necessary).

Oracle Bulk Loading: z/OS Specifics

When you are using bulk loading in the z/OS operating environment, the files used by the SQL*Loader must conform to z/OS data set standards. The data sets can be either sequential data sets or partitioned data sets. Each of the file names supplied to the SQL*Loader are subject to extension and FNA processing.

If you do not specify file names using data set options, then default names in the form of *userid*.SAS.*data-set-extension* apply. The *userid* is the TSO prefix when running under TSO, and it is the PROFILE PREFIX in batch. The *data-set-extensions* are:

BAD for the bad file

CTL for the control file

DAT for the data file

DSC for the discard file

LOG for the log file

If you want to specify file names using data set options, then you must use one of the following forms:

/DD/ddname

/DD/ddname(membername)

Name

For detailed information about these forms, refer to the SQL*Loader chapter in the Oracle user's guide for z/OS.

The SQL*Loader is executed by the Oracle engine by issuing a host-system command from within your SAS session. The data set where the SQLLDR executable resides must be available to your TSO session or allocated to your batch job. Check with your system administrator if you do not know the name or availability of the data set that contains the SQLLDR executable.

On z/OS, the bad file and the discard file are, by default, not created in the same format as the data file. This makes it difficult to load the contents of these files after making corrections. Refer to the section on SQL*Loader file attributes in the SQL*Loader section in the Oracle user's guide for z/OS for information about overcoming this limitation.

Oracle Bulk Loading: Example

The following example shows how to create a SAS data set and use it to create and load to a large Oracle table, FLIGHTS98. This load uses the SQL*Loader direct path method because you specified BULKLOAD=YES. BL_OPTIONS= passes the specified SQL*Loader options to SQL*Loader when it is invoked. In this example, the ERRORS= option enables you to have 899 errors in the load before the load terminates, and the LOAD= option loads the first 5,000 rows of the input data set, SASFLT.FLT98.

```
options yearcutoff=1925;   /* included for Year 2000 compliance */

libname sasflt 'SAS-Data-Library';
libname ora_air oracle user=testuser password=testpass
   path='ora8_flt' schema=statsdiv;

data sasflt.flt98;
   input flight $3. +5 dates date7. +3 depart time5. +2 orig $3.
```

```
      +3 dest $3.  +7 miles +6 boarded +6 capacity;
   format dates date9. depart time5.;
   informat dates date7. depart time5.;
   datalines;
114     01JAN98    7:10  LGA   LAX       2475       172       210
202     01JAN98   10:43  LGA   ORD        740       151       210
219     01JAN98    9:31  LGA   LON       3442       198       250

<...10,000 more observations...>

proc sql;
create table ora_air.flights98
(BULKLOAD=YES BL_OPTIONS='ERRORS=899,LOAD=5000') as
   select * from sasflt.flt98;
quit;
```

During a load, certain SQL*Loader files are created, such as the data, log, and control files. Unless otherwise specified, they are given a default name and written to the current directory. For this example, the default names would be **bl_flights98.dat**, **bl_flights98.log**, and **bl_flights98.ctl**.

Locking in the Oracle Interface

The SAS/ACCESS interface to Oracle supports the following locking options as both LIBNAME and data set options. See the section about the LIBNAME statement in *SAS/ACCESS for Relational Databases: Reference* for additional information about these options.

READ_LOCK_TYPE= NOLOCK | ROW | TABLE
The default value is NOLOCK. The valid values for this option are as follows:

- NOLOCK — table locking is not used during the reading of tables and views.
- ROW — the Oracle ROW SHARE table lock is used during the reading of tables and views.
- TABLE — the Oracle SHARE table lock is used during the reading of tables and views.

Note: If you set READ_LOCK_TYPE= to either TABLE or ROW, you must also set the CONNECTION= option to UNIQUE. If not, an error occurs. △

UPDATE_LOCK_TYPE= NOLOCK | ROW | TABLE
The default value is NOLOCK. The valid values for this option are as follows:

- ROW — the Oracle ROW SHARE table lock is used during the reading of tables and views for update.
- TABLE — the Oracle EXCLUSIVE table lock is used during the reading of tables and views for update.
- NOLOCK — table locking is not used during the reading of tables and views for update.
  - If OR_UPD_NOWHERE=YES, updates are performed using serializable transactions.
  - If OR_UPD_NOWHERE=NO, updates are performed using an extra WHERE clause to ensure that the row has not been updated since it was first read. Updates might fail under these conditions, because other users might modify a row after the row was read for update.

READ_ISOLATION_LEVEL= READCOMMITTED | SERIALIZABLE
Oracle supports the READCOMMITTED and SERIALIZABLE read isolation levels, as defined in the following table. The SPOOL= option overrides the READ_ISOLATION_LEVEL= option. The READ_ISOLATION_LEVEL= option should be rarely needed because the SAS/ACCESS engine chooses the appropriate isolation level based on other locking options.

Table 1.3 Isolation Levels for Oracle

Isolation Level	Definition
SERIALIZABLE	Does not allow dirty reads, non-repeatable reads, or phantom reads.
READCOMMITED	Does not allow dirty reads; does allow non-repeatable reads and phantom reads

UPDATE_ISOLATION_LEVEL= READCOMMITTED | SERIALIZABLE
Oracle supports the READCOMMITTED and SERIALIZABLE isolation levels, as defined in the preceding table, for updates.

This option should be rarely needed because the SAS/ACCESS engine chooses the appropriate isolation level based on other locking options.

Naming Conventions for Oracle

The PRESERVE_COL_NAMES= and PRESERVE_TAB_NAMES= options determine how the interface to Oracle 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.

Oracle objects that can be named include tables, views, columns, and indexes. For the Oracle7 Server, objects also include database triggers, procedures, and stored functions. Use the following Oracle naming conventions:

- A name must start with a letter. However, if the name appears within double quotation marks, it may start with any character.
- A name must be from 1 to 30 characters long, except for database names, which are limited to 8 characters, and link names, which are limited to 128 characters.
- A name may contain the letters A through Z, the digits 0 through 9, the underscore (_), $, and #. If the name appears within double quotation marks, it may contain any characters, except double quotation marks.
- A name is not case-sensitive. For example, CUSTOMER is the same as customer. However, if the name of the object appears within double quotation marks when it is used, then it is case-sensitive.
- A name cannot be an Oracle reserved word.
- A name cannot be the same name as another Oracle object in the same schema.

Data Types for Oracle Servers

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

Note: The SAS/ACCESS interface to Oracle does not support the following Oracle data types: MLSLABEL and ROWID. △

Character Data

CHAR (*n*)
: contains fixed-length character string data with a length of *n*, where *n* must be at least 1 and cannot exceed 255 characters. (The limit is 2,000 characters with an Oracle8 Server.) Note that the Oracle7 Server CHAR data type is not equivalent to the Oracle Version 6 CHAR data type. The Oracle7 Server CHAR data type is new with the Oracle7 Server and uses blank-padded comparison semantics.

LONG
: contains varying-length character string data that is similar to type VARCHAR2. Type LONG is character data of variable length with a maximum length of 2 gigabytes. You can define only one LONG column per table. Available memory considerations might also limit the size of a LONG data type.

VARCHAR2(*n*)
: contains character string data with a length of *n*, where *n* must be at least 1 and cannot exceed 2000 characters. (The limit is 4,000 characters with an Oracle8 Server.) The VARCHAR2 data type is equivalent to the Oracle Version 6 CHAR data type except for the difference in maximum lengths. The VARCHAR2 data type uses nonpadded comparison semantics.

Numeric Data

NUMBER(*p,s*)
: specifies a fixed-point number with an implicit decimal point, where *p* is the total number of digits (precision) and can range from 1 to 38, and *s* is the number of digits to the right of the decimal point (scale) and can range from -84 to 127.

NUMBER(*p*)
: specifies an integer of precision *p* that can range from 1 to 38 and a scale of 0.

NUMBER
: specifies a floating-point number with a precision of 38. A floating-point value can either specify a decimal point anywhere from the first to the last digit or omit the decimal point. A scale value does not apply to floating-point numbers since there is no restriction on the number of digits that can appear after the decimal point.

Other Data Types

DATE
: contains date values. Valid dates are from January 1, 4712 BC to December 31, 4712 AD. The default format is DD-MON-YY, for example ’05-OCT-98’.

LONG RAW
: contains raw binary data of variable length up to 2 gigabytes. Values entered into columns of this type must be inserted as character strings in hexadecimal notation.

RAW(*n*)
: contains raw binary data where *n* must be at least 1 and cannot exceed 255 bytes. (In Oracle Version 8, the limit is 2,000 bytes.) Values entered into columns of this type must be inserted as character strings in hexadecimal notation. You must specify *n* for this data type.

Note: For compatibility with other DBMSs, Oracle supports the syntax for a wide variety of numeric data types, including DECIMAL, INTEGER, REAL, DOUBLE-PRECISION, and SMALLINT. All forms of numeric data types are actually stored in the same internal Oracle NUMBER format. The additional numeric data types are variations of precision and scale. A null scale implies a floating-point number, and a non-null scale implies a fixed-point number. △

Oracle Null and Default Values

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

By default, Oracle columns accept NULL values. However, you can define columns so that they cannot contain NULL data. NOT NULL tells Oracle not to add a row to the table unless the row has a value for that column. When creating an Oracle 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 in *SAS/ACCESS for Relational Databases: Reference*.

Note: To control how SAS missing character values are handled by Oracle, 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 Oracle data types during input operations when you use the LIBNAME statement.

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

Oracle Data Type	Default SAS Format
CHAR(*n*)	$*n*.
VARCHAR2(*n*)	$*n*.
NUMBER	none (BEST. on OS/390)
NUMBER(*p*)	*w*.(BEST. on OS/390)
NUMBER(*p*, *s*)	*w.d*
DATE	DATETIME20.
LONG	$1024.
RAW(*n*)	$HEX*w*.
LONG RAW	$HEX2048.

Oracle data types that are omitted from this table are not supported by SAS/ACCESS.

If Oracle data falls outside valid SAS data ranges, the values are usually counted as missing.

Note: SAS automatically converts Oracle NUMBER types to SAS number formats by using an algorithm that determines the correct scale and precision. When the scale and precision cannot be determined, SAS/ACCESS allows the procedure or application to determine the format.

You can also convert numeric data to character data by using the Pass-Through facility with the Oracle TO_CHAR function. See your Oracle documentation for more details. △

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

Table 1.5 LIBNAME Statement: Default Oracle Data Types for SAS Formats

SAS Variable Format	Oracle Data Type
$*w*.	VARCHAR2(*n*)***
w. with SAS format name of NULL	NUMBER(*p*)
w.d with SAS format name of NULL	NUMBER(*p,s*)
all other numerics *	NUMBER (NUMBER(38,10) on OS/390)
datetime*w.d*	DATE
date*w*.	DATE
time. **	DATE

* Includes all SAS numeric formats, such as BINARY8 and E10.0.
** Includes all SAS time formats, such as TOD*w,d* and HHMM*w,d*.
***If the SAS char variable format is greater than $4000, the default Oracle data type is LONG. If you use Oracle7 and the CHAR variable format is between $2000. and $4000. then use the DBTYPE= option to change the default VARCHAR2 type to LONG.

To override these data types, use the DBTYPE= data set option during output processing.

ACCESS Procedure Data Conversions

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

Table 1.6 PROC ACCESS: Default SAS Formats for Oracle Data Types

Oracle Data Type	Default SAS Format
CHAR(*n*)	$*n*. (*n* <= 200) $200. (*n* > 200)
VARCHAR2(*n*)	$*n*. (*n* <= 200) $200. (*n* > 200)
FLOAT	BEST22.
NUMBER	BEST22.
NUMBER(*p*)	*w*.
NUMBER(*p*, *s*)	*w.d*
DATE	DATETIME16.
LONG	$200.

Oracle Data Type	Default SAS Format
RAW(*n*)	$*n*. (*n* < 200) $200. (*n* > 200)
LONG RAW	$200.

Oracle data types that are omitted from this table are not supported by SAS/ACCESS. If Oracle data falls outside valid SAS data ranges, the values are usually counted as missing.

The following table shows the correlation between the Oracle NUMBER data types and the default SAS formats that are created from that data type.

Table 1.7 Default SAS Formats for Oracle NUMBER Data Types

Oracle NUMBER Data Type	Rules	Default SAS Format
NUMBER(*p*)	0 < *p* <= 32	(*p* + 1).0
NUMBER(*p*,*s*)	*p* > 0, *s* < 0, |*s*| < *p*	(*p* + |*s*| + 1).0
NUMBER(*p*,*s*)	*p* > 0, *s* < 0, |*s*| >= *p*	(*p* + |*s*| + 1).0
NUMBER(*p*,*s*)	*p* > 0, *s* > 0, *s* < *p*	(*p* + 2).*s*
NUMBER(*p*,*s*)	*p* > 0, *s* > 0, *s* >= *p*	(*s* + 3).*s*
NUMBER(*p*)	*p* > 32	BEST22. SAS selects format
NUMBER	*p*, *s* unspecified	BEST22. SAS selects format

Note: The general form of an Oracle number is NUMBER(*p,s*) where *p* is the precision and *s* is the scale of the number. Oracle defines precision as the total number of digits, with a valid range of -84 to 127. However, a negative scale means that the number is rounded to the specified number of places to the left of the decimal. For example, if the number 1,234.56 is specified as data type NUMBER(8,-2), it is rounded to the nearest hundred and stored as 1,200. △

DBLOAD Procedure Data Conversions

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

Table 1.8 PROC DBLOAD: Default Oracle Data Types for SAS Formats

SAS Variable Format	Oracle Data Type
$*w*.	CHAR(*n*)
w.	NUMBER(*p*)
w.d	NUMBER(*p,s*)
all other numerics *	NUMBER
datetime*w.d*	DATE
date*w*.	DATE
time. **	NUMBER

* Includes all SAS numeric formats, such as BINARY8 and E10.0.

** Includes all SAS time formats, such as TOD*w,d* and HHMM*w,d*.