ABSTRACT
This paper reviews the ORACLE LIBNAME engine and shows how to use it to extract data from an ORACLE database. Common parameters for the ORACLE LIBNAME engine are illustrated. We discuss issues related to reporting resource usage. ORACLE hints are described and explained as a way to reduce the time it takes to extract data from an ORACLE table and to improve processing. Several examples of ORACLE hints are given. The paper discusses the use of ORACLE indexes as a way to improve efficiency and avoiding sorting. Examples are given of the use of the DBKEY option to efficiently extract part of an ORACLE table.

INTRODUCTION
The purpose of this paper is to discuss using the ORACLE LIBNAME engine to extract data from an ORACLE database. The U.S. Census Bureau maintains its Business Register in an ORACLE database. This ORACLE database is used for its ability to store, process, and retrieve data. Access to the database, however, is provided through a series of consolidated SAS® datasets. Consolidated SAS data combine data from an ORACLE parent table with information from multiple child tables. These SAS datasets are the primary access to the Business Register for Business Register users. A series of eight consolidated SAS datasets are produced monthly. Extracting the data is a key part of creating these monthly SAS datasets. Doing it efficiently is important to minimizing the resources required to do it. This paper discusses how we use the ORACLE LIBNAME engine and other features of SAS to do this.

ORACLE LIBNAME ENGINE
The ORACLE LIBNAME engine allows SAS users to access ORACLE tables in the same way they access SAS datasets. All tables in the database (or a part of the database called a schema) are treated like SAS datasets in a defined library. Once the ORACLE LIBNAME engine is invoke, you can work with ORACLE tables just like you would with SAS datasets in any SAS library.

The code below uses the ORACLE LIBNAME engine to access ORACLE tables. Macro variables are used because SAS MPCONNECT is used extensively to access the same ORACLE database in multiple child jobs. Using macro variables allows us to enter the database names and passwords at one time. This reduces most of the work when a database name, user, or password needs to be changed. Putting the macro variables into the “autoexec.sas” file, allows us to set parameters one time in one place. This is important when moving between test and production machines, changing passwords, and in other situations. The PUT statements at the end allow you to verify the code used.

```sas
options sastrace=',,,d' sastraceloc=SASLOG;

%LET DATABASE   =  BRSAS  ;
%LET PATH       =  BRSAS  ;
%LET SCHEMA     =  BR_OWNER  ;
%LET USER       =  ops$chapm001  ;
%LET PWRD       =  password  ;

LIBNAME &DATABASE ORACLE path=&path SCHEMA=&SCHEMA USER=&USER orapw=&PWRD access=readonly dbindex=YES read_lock_type=nolock readbuff=250;
```
When the ORACLE LIBNAME engine is used, ORACLE tables in the ORACLE database appear just like SAS datasets in a SAS library. The same skills used to manage and manipulated ordinary files in a SAS library can be used to manage ORACLE tables.

To disassociate an ORACLE database from a SAS LIBNAME, the “clear” option can be used.

```
LIBNAME &DATABASE CLEAR ; RUN;
```

The name of ORACLE database associated with a SASLIBNAME can be identified using the LIST option.

```
LIBNAME &DATABASE LIST; RUN;
```

**ALTERNATIVES**

There are at least two alternatives to using the ORACLE LIBNAME engine to extract data from an ORACLE database. One is SAS/ACCESS; the other is the SQL Procedure Pass-Through facility. SAS/ACCESS requires the definition of descriptors (that describe the table being extracted) and views (that define what information is to be extracted). Some consider them more complicated than either the ORACLE LIBNAME engine or the SQL Pass-through. Proc SQL pass-through does many of the same things as the ORACLE LIBNAME engine. It may be more suited for people who prefer programming in SQL. The ORACLE LIBNAME engine, SAS/ACCESS, and the SAS SQL Pass-Through facility are described in Chapter 7, Advanced Topics in SAS/ACCESS (SAS Institute Inc., 1999b).

**SYSTEM OPTIONS**

Two SAS options that are useful with the ORACLE LIBNAME engine are the SASTRACE and SASLOG options. This is documented in Chapter 5 “Macro Variables and System Options” of SAS/ACCESS Software for Relational Databases: Reference, Version 8 (SAS Institute, 1999b).

- Option SASTRACE=',,,d'

The SASTRACE option provides information on the SQL commands that SAS sends to ORACLE. When you invoke this option, the system echoes the SQL code that is sent to the ORACLE database. This allows you to see what SAS is asking ORACLE to do.

- Option SASTRACELOC= SASLOG (FILENAME)

The SASTRACELoc option tells SAS when to store the SQL code the it sends to ORACLE. The default is to write the code to the SAS LOG. Specifying a location other than the LOG allows you to keep your LOG clear. The SASTRACE options can write a lot of dense lines to your LOG. An example of the SASTRACE output is given below.

```
109 options sastrace=',,,d' sastraceloc=saslog ;run
```
Information from the SASTRACE option is useful in diagnosing and solving problems. This is illustrated in a discussion of the use of DBKEY option with an ORACLE table.

The code below uses the ORACLE LIBNAME engine to list the ORACLE tables in the library (ORACLE database) and to look at the variables and their attributes for one table in the ORACLE database. Notice that they do not contain the all the information that is received when you use them with SAS datasets. PROC CONTENTS does not show the number of observations, indexes, and creation and modification dates.

NOTE: Remote submit to EPBA53 commencing.

```sas
49 proc datasets library=brsas;
    -----Directory-----
    Libref:                 BRSAS
    Engine:                 ORACLE
    Access:                 READONLY
    Physical Name:          BRSAS
    Schema/User:            BR_OWNER

    #  Name                         Memtype
    ------------------------------
    1  ACCESS_DIARIES               DATA
    2  ACCT_MGT                     DATA
{entries deleted to save space}
    108  TEMP_MSD1                    DATA
    109  USERS                        DATA

49 !         quit;
```

NOTE: PROCEDURE DATASETS used:
real time 0.13 seconds
cpu time 0.02 seconds

```sas
50 proc contents data=brsas.links;run;
```

NOTE: PROCEDURE CONTENTS used:
real time 0.05 seconds
cpu time 0.02 seconds

NOTE: The PROCEDURE CONTENTS printed page 5.
NOTE: Remote submit to EPBA53 complete.
MACRO VARIABLES

Specifying parameters that can change as macro variables can save considerable work. If the same ORACLE LIBNAME engine code is used in multiple programs. This allows you to define variables such as database name, user name, and use passwords one place.

```sas
%LET DATABASE   =  BRSAS  ;
%LET PATH       =  BRSAS  ;
%LET SCHEMA     =  BR_OWNER  ;
%LET USER       =  ops$chapm001  ;
%LET PWRD       =  password  ;
```

Hard coding these parameters can result in a lot of work when you shift between a production and test database and change users responsible for maintaining the code. These macro variables can be stored in the “autoexec.sas” file or special macro.

SAS/ACCESS ORACLE ENGINE CONNECTION OPTIONS

There are a variety of options that can be used to specify and modify the ORACLE LIBNAME engine. Selected options related to reading ORACLE databases are discussed here.

**Connection Options** – A group of LIBNAME parameters are associated with connecting to the ORACLE database to be associated with SAS LIBNAME. The four most common are:

**USER**: ORACLE user name (e.g. ops$chapm001)

**PASSWORD**: The ORACLE password associated with the ORACLE user name.

**PATH**: This is the name of the Oracle database.

**SCHEMA**: Within a database, a set of tables, indexes, and other ORACLE objects may be collected together and associated with a schema name. When a schema is specified, only that group of tables and objects will be used.

Prompting for User Name and Password -- The option “DBPOMPT = YES” can be used when you do not want or security procedures to not allow you to hard code a user name or password into SAS code. When this option is used, the SAS displays a prompting dialog box that allows you to enter the database name, user name, and password instead of entering them with the ORACLE LIBNAME statement. This may be great for interactive applications; but, may not really be suitable to batch applications.

SAS/ACCESS ORACLE LIBNAME OPTIONS

There are a variety of options that can be used to specify and modify the ORACLE LIBNAME engine. Selected options related to reading ORACLE databases are discussed here. When an option is specified is applies to all the ORACLE tables associated with SAS LIBNAME.

**DBINDEX**: The option “DBINDEX=YES” tells SAS to contact ORACLE to determine if there is an ORACLE index
associated with an ORACLE variable. When you use a “BY” statement with an ORACLE table, ORACLE index, the data will be returned in INDEX order.

**DBPROMPT**: The option “DBPROMPT = YES” can be used when you do not want or security procedures to not allow you to hard code a user name or password into SAS code. When this option is used, the SAS displays a prompting dialog box that allows you to enter the database name, user name, and password instead of entering them with the ORACLE LIBNAME statement.

**PRESERVE_COL_NAMES**: ORACLE allows variables to contain characters that are not permitted in SAS names. An example is the “$” character. The options “PRESERVE_COL_NAMES=YES” allow the SAS user to preserve these ORACLE names not supported by SAS.

**PRESERVE_TAB_NAMES**: Similar to PERSERVE_COL_NAMES, ORACLE allows tables to contain characters that are not permitted in SAS names. An example is the “$” character. The options “PRESERVE_TAB_NAMES=YES” allows you to preserve these ORACLE names not supported by SAS.

**READBUFF**: This option tells SAS how many records to read from ORACLE at one time. The default for SAS 8.2 is 250. For earlier versions, it was 25. For versions of SAS before 8.2, the READBUFF parameter should be set to at least 250 to improve efficiency.

There are other options associated with the ORACLE LIBNAME engine that are primarily using with writing information to ORACLE tables. They are not discussed here.

**MEASURING AND COMPARING PERFORMANCE**

Measuring and comparing the performance of the same code over time or for two sets of the code that do the same thing can be a difficult process. It is made more difficult when it is done on a multi-user system. Here we discuss measuring performance of SAS code.

**SAS**

The primary way to collect information on performance is using the FULLSTIMER AND STIMER options. Statistics reported by the STIMER option is a subset of the information reported under FULLSTIMER. The information reported using the option is illustrated below.

```
13   options nofullstimer nostimer;run;
14   proc univariate data=test;
15   var x;
16   run;
17   options nofullstimer stimer;run;
18   proc univariate data=test;
19   var x;
20   run;
21   options fullstimer stimer;run;
22   proc univariate data=test;
23   var x;
24   run;
```

NOTE: PROCEDURE UNIVARIATE used:
real time           0.28 seconds
cpu time            0.21 seconds

NOTE: PROCEDURE UNIVARIATE used:
real time           0.25 seconds
user cpu time       0.19 seconds
system cpu time     0.01 seconds
Memory              3287k
A detailed explanation of the information provided by the FULLSTIMER option is provided in the Scalability and Performance Community on the SAS Institute Web site (http://support.sas.com/md/scalability/tools/fullstim/fullstim.html). Two key measures are real time and cpu time. Real time is the actual time it takes to do the procedure; CPU time is the amount of CPU it takes to do it. In windows CPU time is divided between user and system CBPU time. “user cpu time” measures the cpu time associated with the SAS code; “system cpu time” measures the cpu time associated with other system processes associated with job. Some operating system like OpenVMS reports a single “CPU TIME” and does not divide it into two separate parts.

An alternative to using information for the FULLSTIMER option would be to collect the information using the Application Response Measurement (ARM) SAS macros. These macros are provided with the SAS system and are describe in the Scalability and Performance Community (http://support.sas.com/md/scalability/tools/arm/arm.html). While the ARM Macros appear to be aimed at managing the performance of interactive distributed applications, they are useful with batch applications.

MULTI-USER SYSTEM

Comparing performance is more complicated when working on a multi-user system. Other users will affect the performance on the ORACLE database; other users on the system will affect the general performance of any job. This occurs because the operating system attempts to switch between different programs and some critical jobs can be given higher priority to resources. The bottom line is the resources available to a single job with multiple steps or multiple jobs will not have the same resources allocated. It is not always clear when the resources change. There may be wide fluctuations in performance in multiple runs of the same job.

ORACLE HINTS

ORACLE hints are part of the ORACLE database. They are NOT something developed by SAS or incorporated into SAS. SAS allows the ORACLE LIBNAME engine to make use of these ORACLE hints so that SAS users can have more control over how they work ORACLE.

ORACLE contains an optimizer that makes decisions on how to handle different situations. Examples are when to use an index and when multiple indexes exist which index to use. The optimizer makes these decisions based on the statistics collected on the database. Statistics are collected using the ANALYZE feature of ORACLE which must be run to be kept up-to-date. When an ANALYZE is run on an ORACLE database, it can affect performance. When the statistics are out-of-date or there are special circumstances, ORACLE hints can be used to guide ORACLE in making decisions.

ORACLE SQL Hints are several types:

- Hints for Optimization
- Hints for Access
- Hints of Join Order
- Hints for Joint Operations
- Hints for Parallel Execution

Hints discussed here include hints for optimization, access, and parallel execution. ORACLE SQL Hints can be used in both the data step and PROCs. ORACLE hints are not parameters set when the ORACLE LIBNAME is defined. Oracle hints are specified when the ORACLE table is used. They can be used either in the DATA step or a PROC. Two examples are given below.

```sas
DATA BSR_SMPL;
SET BRSAS.EINUNIT_BSR_SMPL;  
(ORHINTS='/*+ FULL(EINUNIT_BSR_SMPL) PARALLEL (einunit_bsr_smpl , 5) */');
RUN;

PROC FREQ DATA=BRSAS.EMPLOYER_UNITS; 
(ORHINTS='/*+ all_rows */');
table empunit_typ/missing;run;
```

The format of the Oracle hint is important. If any of the rules are violated, ORACLE considers the hint to be a comment and ignores it.

COMMON HINTS
ORACLE documentation discussed the details of ORACLE hints. Three common hints are given below:

ALL_ROWS – The ALL_ROWS hint chooses a cost-based approach to reduce the total resource to read all the records in the table. The hint has the following form:

```sas
DATA WORK.EMPLOYER_UNITS;
SET BRSAS.EMPLOYER_UNITS
   (ORHINTS='/*+ ALL_ROWS */);
RUN;
```

INDEX – The INDEX hint tells ORACLE to use a specific hint when extracting data from an ORACLE table. The hint has the following form:

```sas
DATA WORK.EMPLOYER_UNITS;
SET BRSAS.EMPLOYER_UNITS
   (ORHINTS='/*+ INDEX(EMPLOYER_UNITS, EU_PK) */');
BY EMPUNIT_ID;
RUN;
```

The hint specifies both the table of the index refers to as well as the specific index name. A table can have multiple indexes. The section below describes how to identify what indexes there are as well as what variables they use.

PARALLEL -- The PARALLEL hint tells ORACLE to use multiple processors to accomplish the job. The hint is of the form:

```sas
DATA WORK.EMPLOYER_UNITS;
SET BRSAS.EMPLOYER_UNITS
   (ORHINTS='/*+ PARALLEL (EMPLOYER_UNITS, 8) */');
RUN;
```

The hint requires the ORACLE table be specified as well as the number of processors to use.

**ORACLE INDEXES**

Indexes are an invaluable help in processing with ORACLE tables. When an index to an ORACLE table can be used, it can result in significant savings in both processing time and processing resources. An index is an alternative to sorting data and in retrieving a subset of data.

Most, if not all, ORACLE tables have records stored in the order they are received. ORACLE indexes are used to specify the order records are extracted and to improve the response time when the table is queried. ORACLE tables can have multiple indexes for a table. ORACLE is rather intelligent in identifying which index to use. You can however use ORACLE Hints when necessary to specify exactly which index to use. This section discusses how to identify what indexes are available, and how to use them.

**USING THE INDEX**

When no index is used, the data is retrieved in the order it is stored in the ORACLE table. The order can be altered with a "BY" statement. The time is takes depends on whether the "BY" variables have an ORACLE Index. Using the ORACLE Index will eliminate the need to sort the file when an index exists and will improve performance if variables in the index can by used in a where statement. The use of the index can control whether the records in an ORACLE table are extracted in their natural order, ascending order of the index variables, or descending order of the index variables. This is illustrated below. The table EMPLOYER_UNITS has an index that identifies EMPUNIT_ID as the index variable.

The code below extracts data in the order they were entered into the ORACLE table to a SAS dataset name UNORDERED. When the index exists and a BY statement is used, the data is extracted in ascending order by the value of the index.

```sas
105  data UNORDERED;
106  SET BRSAS.EMPLOYER_UNITS;
107  RUN;
```

NOTE: There were 14160575 observations read from the data set BRSAS.EMPLOYER_UNITS.
NOTE: The data set WORK.UNORDERED has 14160575 observations and 50 variables.
NOTE: DATA statement used:
      real time        32:12.06
PROC SORT DATA=UNORDERED; BY EMPUNIT_ID; RUN;

NOTE: There were 14160575 observations read from the data set WORK.UNORDERED.
NOTE: The data set WORK.UNORDERED has 14160575 observations and 50 variables.
NOTE: PROCEDURE SORT used:
real time  32:40.20
cpu time   15:59.10

data ASCENDING_ORDER;
SET BRSAS.EMPLOYER_UNITS ;
BY EMPUNIT_ID;
RUN;

NOTE: There were 14160575 observations read from the data set BRSAS.EMPLOYER_UNITS.
NOTE: The data set WORK.ASCENDING_ORDER has 14160575 observations and 50 variables.
NOTE: DATA statement used:
real time  39:15.26
cpu time   24:58.43

Comparison of Using ORACLE Index to Sorting By SAS

<table>
<thead>
<tr>
<th>Measures</th>
<th>No Index + Sort</th>
<th>Use ORACLE Index</th>
<th>DIFFERENCE</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>ABSOLUTE</td>
</tr>
<tr>
<td>Real Time</td>
<td>64:86</td>
<td>39.25</td>
<td>- 25.61</td>
</tr>
<tr>
<td>CPU Time</td>
<td>38.60</td>
<td>24.97</td>
<td>-13.63</td>
</tr>
</tbody>
</table>

The dataset UNORDERED is sorted in the order it was loaded into the ORACLE table. The dataset ASCENDING_ORDER is sorted with the smallest value of EMPUNIT_ID first; the dataset DESCENDING_ORDER is sorted with the largest value of EMPUNIT_ID first and the smallest value last.

Without using the Index, the UNORDERED SAS dataset would have had to be sorted before it was comparable to the ASCENDING_ORDER SAS dataset. The gain from using the index depends on the natural ordering of the ORACLE table by the indexed variables.

IDENTIFYING AVAILABLE INDEXES

It is not always apparent what indexes are available for a table in an ORACLE database. Often it is difficult to know what indexes are available for a particular table. Information – similar to SAS dictionary files – on ORACLE indexes is available from the ORACLE database. This information is stored in the schema “SYS” in a table called “ALL_IND_COLUMNS”. The code below prints a list of all indexes associated with all tables in the schema “BR_OWNER” in the database BRSAS. The list is sorted by table, index name, and the variables in the order they are used in the index. You may need to contact your ORACLE DBA for permission to access these tables.

The code below prints the contents of the ORACLE table ALL_IND_COLUMNS in the SYS schema that contains the names of all indexes associated with each table and the variables that make up the index. The code below uses PROC REPORT to display the tables and their respective indexes and variables.

%LET DATABASE = SYS ;
%LET PATH     = BRSAS ;
%LET SCHEMA   =  SYS   ;
%LET USER     =  ops$chapm001 ;
%LET PWRD     =  april2005 ;
LIBNAME &DATABASE ORACLE
    path    = "&path"
    USER    = "&user"
    SCHEMA  = "&SCHEMA"
    USER    = "&user"
    orapw   = "&pwrd"
    access=readonly
    dbindex=YES
    read_lock_type=nolock
    readbuff=250;
RUN;

PROC REPORT DATA=SYS.ALL_IND_COLUMNS (WHERE=(TABLE_OWNER="BR_OWNER")) NOWD ;
COLUMN TABLE_NAME INDEX_NAME
   ( "_COLUMN_ " COLUMN_NAME COLUMN_POSITION COLUMN_LENGTH CHAR_LENGTH DESCEND);
DEFINE TABLE_NAME /GROUP FORMAT=$20.;
DEFINE INDEX_NAME /GROUP FORMAT=$20.;
DEFINE COLUMN_NAME /DISPLAY LEFT FORMAT=$20. 'NAME';
DEFINE COLUMN_POSITION /DISPLAY CENTER FORMAT=9. 'POSITION';
DEFINE COLUMN_LENGTH /DISPLAY CENTER FORMAT=7. 'LENGTH';
DEFINE CHAR_LENGTH /DISPLAY CENTER FORMAT=11. 'CHARACTERS';
DEFINE DESCEND /DISPLAY CENTER FORMAT=$13.;
BREAK AFTER TABLE_NAME/ SKIP;
BREAK AFTER INDEX_NAME/ SKIP;
RUN;
The contents of the ORACLE table containing information on the index is given below.

<table>
<thead>
<tr>
<th>#</th>
<th>Variable</th>
<th>Type</th>
<th>Len</th>
<th>Pos</th>
<th>Format</th>
<th>Informat</th>
<th>Label</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>CHAR_LENGTH</td>
<td>Num</td>
<td>8</td>
<td>4144</td>
<td></td>
<td>$4000.</td>
<td>CHAR_LENGTH</td>
</tr>
<tr>
<td>7</td>
<td>COLUMN_LENGTH</td>
<td>Num</td>
<td>8</td>
<td>4136</td>
<td></td>
<td>$4000.</td>
<td>COLUMN_LENGTH</td>
</tr>
<tr>
<td>5</td>
<td>COLUMN_NAME</td>
<td>Char</td>
<td>4000</td>
<td>128</td>
<td>$30.</td>
<td>$30.</td>
<td>COLUMN_NAME</td>
</tr>
<tr>
<td>6</td>
<td>COLUMN_POSITION</td>
<td>Num</td>
<td>8</td>
<td>4128</td>
<td></td>
<td>$30.</td>
<td>COLUMN_POSITION</td>
</tr>
<tr>
<td>9</td>
<td>DESCEND</td>
<td>Char</td>
<td>4</td>
<td>4152</td>
<td>$4.</td>
<td>$4.</td>
<td>DESCEND</td>
</tr>
<tr>
<td>2</td>
<td>INDEX_NAME</td>
<td>Char</td>
<td>30</td>
<td>96</td>
<td>$30.</td>
<td>$30.</td>
<td>INDEX_NAME</td>
</tr>
<tr>
<td>1</td>
<td>INDEX_OWNER</td>
<td>Char</td>
<td>30</td>
<td>0</td>
<td>$30.</td>
<td>$30.</td>
<td>INDEX_OWNER</td>
</tr>
<tr>
<td>4</td>
<td>TABLE_NAME</td>
<td>Char</td>
<td>30</td>
<td>64</td>
<td>$30.</td>
<td>$30.</td>
<td>TABLE_NAME</td>
</tr>
<tr>
<td>3</td>
<td>TABLE_OWNER</td>
<td>Char</td>
<td>30</td>
<td>64</td>
<td>$30.</td>
<td>$30.</td>
<td>TABLE_OWNER</td>
</tr>
</tbody>
</table>

A PROC REPORT table below shows the indexes available for selected ORACLE tables, the names of the index,
and order of the variables used by the index.

<table>
<thead>
<tr>
<th>COLUMN</th>
</tr>
</thead>
<tbody>
<tr>
<td>TABLE_NAME</td>
</tr>
<tr>
<td>INDEX_NAME</td>
</tr>
<tr>
<td>NAME</td>
</tr>
<tr>
<td>POSITION</td>
</tr>
<tr>
<td>LENGTH</td>
</tr>
<tr>
<td>CHARACTERS</td>
</tr>
<tr>
<td>DESCEND</td>
</tr>
</tbody>
</table>
The information above identifies indexes that are available for an ORACLE table, what variables they contain, and the order of the index. It is the basis for using BY statements and certain ORACLE Hints. Notice that many tables contain more than one index. Each index often contains more than one variable. The notation PK in an index names refers to the variables being a primary key for an ORACLE table; the notation IDX refers an index that is not a primary key.

**EXTRACTING PART OF AN ORACLE TABLE – “ORACLE DBKEY”**

Sometimes only part of an ORACLE table needs to be extracted and put into a SAS dataset. If the criteria used to identify the records to be extracted are based on an ORACLE index for the table, the “DBKEY=” option may improve the speed of the extraction over that of a MERGE. This option is similar to the “KEY = “ option used with two SAS data sets. For SAS datasets this is more efficient because, when there are relatively few records to extract, the index allows SAS to go directly to each individual record without reading through the entire ORACLE table.

**NO INDEX**

When there is no index, the “DBKEY=” option cannot be used. The ORACLE data must be extracted to a SAS data set and sorted or indexed. Once it is sorted, the final dataset would be created using a MERGE. If the final dataset is indexed, a “KEY=” option can be used.

**INDEX – MERGE - NO DBKEY**

When there is an ORACLE index, it is common to use the merge statement to extract part of the data. You must first extract data from the ORACLE table and then either index or sort the SAS dataset before merging it with the sample dataset.

The code to do the merge is given below followed by the LOG. The LIBANME BRSAS is a LIBNAME created using the ORACLE LIBNAME engine. As illustrated below, the effect of this code is to have ORACLE extract and sort the data.
INDEX – DBKEY

When there is an ORACLE index you can also use the DBKEY option. The source code to do that is given below. The effect of this code is to have ORACLE only extract the records in the sample file. This can be seen because of the “WHERE” statement that SAS sends to ORACLE to extract the data.

NOTE: Remote submit to EPBA53 commencing.

NOTE: There were 707491 observations read from the data set WORK.SAMPLE.
NOTE: There were 14160575 observations read from the data set BRSAS.EMPLOYER_UNITS.
NOTE: The data set WORK.SAMPLE1 has 707491 observations and 50 variables.
NOTE: DATA statement used:
  real time           23:42.29
  cpu time            20:42.57
NOTE: Remote submit to EPBA53 complete.
COMPARISON: “DBKEY=” VERSUS “MERGE”

Review of the output of the SASTRACE option shows that the commands sent to ORACLE are basically different. With the merge statement ORACLE returns the entire file sorted by the index. With the DBKEY statement, ORACLE returns only the records associated with the sample file. When you look at the real and cpu times used, it appears to have taken almost the same real time; but, the cpu time is significantly less when the DBKEY statement is used. This is likely due to smaller number of records returned and not having to sort the complete universe.

<table>
<thead>
<tr>
<th>Measures</th>
<th>DBKEY</th>
<th>MERGE</th>
<th>DIFFERENCE</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ABS</td>
<td>RELATIVE</td>
<td></td>
</tr>
<tr>
<td>Real Time</td>
<td>22.97</td>
<td>23.70</td>
<td>-1.3</td>
</tr>
<tr>
<td></td>
<td>-3.0%</td>
<td>-46.9%</td>
<td></td>
</tr>
<tr>
<td>CPU Time</td>
<td>11.00</td>
<td>20.70</td>
<td>-9.7</td>
</tr>
<tr>
<td></td>
<td>-46.9%</td>
<td>-46.9%</td>
<td></td>
</tr>
</tbody>
</table>

All computing done on COMPAQ Alpha under OpenVMS

CONCLUSIONS

The ORACLE LIBNAME engine is a valuable way to extract data from an ORACLE database. It is particularly useful for SAS programmers that do not work with SQL. The effectiveness and efficiency depends in part in knowing what parameters to use. Using existing ORACLE indexes may save significant resources when extracting data. Use of the index can avoid the need for sorting the data in SAS. It can reduce resource used to extract data when an index variable is used in a where statement or used together with the DBKEY option. ORACLE hints can be used in conjunction with SAS to extract data with multiple processors to use or not use a specific index.

REFERENCES


Oracle Corporation. Oracle 8i Designing and Tuning for Performance Release 2 (8.1.1) Part Number A76992-01.

ACKNOWLEDGMENTS

SAS is a Registered Trademark of the SAS Institute, Inc. of Cary, North Carolina.

DISCLAIMER

This paper reports the results of research and analysis undertaken by Census Bureau staff. It has undergone a more limited review by the Census Bureau then its official publications. This report is released to inform interested parties and to encourage discussion.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the authors at:

David D. Chapman
Economic Planning and Coordination Division
Selvaratnam Sridharma  
Economic Planning and Coordination Division  
U.S. Bureau of the Census  
Address  
Washington, DC, 20233-6100  
301-763-6774  
Email: selvaratnam.sridharma@census.gov