Finding Oracle® Table Metadata: When PROC CONTENTS Is Not Enough

Jeremy W. Poling, B&W Y-12 L.L.C., Oak Ridge, TN

ABSTRACT

Complex Oracle® databases often contain hundreds of linked tables. For SAS/ACCESS® Interface to Oracle software users who are unfamiliar with a database, finding the data they need and extracting it efficiently can be a daunting task. For this reason, a tool that extracts and summarizes database metadata can be invaluable. This paper describes how Oracle table metadata can be extracted from the Oracle data dictionary using the SAS/ACCESS LIBNAME statement in conjunction with traditional SAS® DATA step programming techniques. For a given Oracle table, we discuss how to identify the number of observations and variables in the table, variable names and attributes, constraints, indexes, and linked tables. A macro is presented which can be used to extract all the aforementioned metadata for an Oracle table and produce an HTML report. Once stored as an autocall macro, the macro can be used to quickly identify helpful information about an Oracle table that cannot be seen from the output of PROC CONTENTS.

This paper assumes that the reader has a basic understanding of DATA step programming, the macro language, and the SAS/ACCESS LIBNAME statement. A basic understanding of relational database management system concepts is also helpful, but not required.

INTRODUCTION

Metadata is typically defined simply as “data about data.” For any SAS data set, a programmer can view useful metadata simply by using PROC CONTENTS. But, what if the table of interest resides in an Oracle database? Unfortunately, finding Oracle table metadata is not quite as straightforward as finding SAS data set metadata.

This paper will explain how to extract metadata from the Oracle data dictionary using the SAS/ACCESS LIBNAME statement. The Oracle data dictionary consists of a set of tables that store Oracle metadata. This paper will describe only a small fraction of the metadata contained in the data dictionary. Once we have identified the Oracle table metadata that we find useful, we can write a SAS autocall macro that can be called upon to quickly extract the metadata and generate a report.

An Oracle database is composed of multiple schemas. A schema is simply a set of database objects, such as tables, views and indexes, which are associated with a particular owner. For example purposes, the SAS codes presented in this paper reference a table named **purchases** that resides in the **sch** schema of a fictional Oracle database.

To avoid confusion associated with differing terminology, this paper will refer to the records of both Oracle tables and SAS data sets as “observations.” Furthermore, we will refer to the fields of both Oracle tables and SAS data sets as “variables.”

FINDING ORACLE TABLE METADATA WITH PROC CONTENTS

You can, of course, specify a SAS/ACCESS view of an Oracle table in the DATA= option of a PROC CONTENTS step. An example of a typical SAS/ACCESS LIBNAME statement and a PROC CONTENTS step follows. The output generated by the CONTENTS procedure is shown in Figure 1.

```
 libname oralib oracle user="userID" password="xxxxxx" path="dbname" schema="sch";

proc contents data=oralib.purchases;
run;
```

```
                             The CONTENTS Procedure

       Data Set Name        ORALIB.PURCHASES     Observations          .
       Member Type          DATA                 Variables             7
       Engine               ORACLE               Indexes               0
       Created              .                    Observation Length    0
       Last Modified        .                    Deleted Observations  0
       Protection                                Compressed            NO
       Data Set Type                             Sorted                NO
       Label
       Data Representation  Default
       Encoding             Default

                  Alphabetic List of Variables and Attributes

   # Variable                Type Len Format      Informat    Label

   4 CUSTOMER_ID             Char  12 $12.        $12.        CUSTOMER_ID
   5 DATE_PURCHASE           Num   12 DATETIME20. DATETIME20. DATE_PURCHASE
   3 EMPLOYEE_ID             Char  12 $12.        $12.        EMPLOYEE_ID
   2 PURCHASE_CODE           Char   1 $1.         $1.         PURCHASE_CODE
   7 PURCHASE_COMMENTS       Char  12 $12.        $12.        PURCHASE_COMMENTS
   1 PURCHASE_ORDER_ID       Char  12 $12.        $12.        PURCHASE_ORDER_ID
   6 TOT_PAYMENT_DUE_AMOUNT  Num    8 12.2        12.2        TOT_PAYMENT_DUE_AMOUNT
```

Figure 1: PROC CONTENTS output produced on a SAS/ACCESS view of an Oracle table

The PROC CONTENTS output will tell you the number of variables in the Oracle table with their names, types, lengths, and formats. However, there is no additional information in the PROC CONTENTS output that is useful. For example, PROC CONTENTS does not identify the number of observations in the Oracle table, Oracle indexes, or Oracle integrity constraints. Even though the output indicates that there are zero SAS indexes defined, one or more Oracle indexes could be defined for the table. Other traditional methods for obtaining SAS data set metadata, such as querying the SAS dictionary tables with PROC SQL, produce similar results. All of this missing information can be crucial for the SAS programmer when developing applications that access the Oracle data. Fortunately, all of the missing pieces can be obtained if we know the right places to look.

USING ORACLE DATA DICTIONARY VIEWS

The Oracle data dictionary is a set of read-only tables that contain Oracle database metadata. Views of the Oracle data dictionary tables are stored in the SYS schema. Provided that your Oracle database administrator has given you the appropriate permissions, you can use SAS/ACCESS Interface to Oracle software to connect to the database and access the data dictionary views via a LIBNAME statement. For example, the LIBNAME statement below creates a SAS library named **orasys** that contains views of an Oracle database's data dictionary.

```
  libname orasys oracle user="userID" password="xxxxxx" path="dbname" schema="sys";
```

After the LIBNAME statement has executed, the user can access data stored in hundreds of data dictionary views using SAS. The data dictionary views with a prefix of "ALL" contain information pertaining to the user's perspective of the database. These views often contain metadata of interest to a SAS programmer. The author has found the following five data dictionary views to be especially useful: ALL_TABLES, ALL_TAB_COLUMNS, ALL_CONSTRAINTS, ALL_CONS_COLUMNS, and ALL_IND_COLUMNS. Each of these views includes a variable named TABLE_NAME that denotes the Oracle table name and a variable named OWNER that denotes the schema, except for the ALL_IND_COLUMNS view where the schema is identified by the variable named TABLE_OWNER. We will briefly discuss each of these data dictionary views in this paper. The reader is encouraged to consult the *Oracle Database Reference* for details pertaining to other Oracle data dictionary views.

As an alternative to the SAS/ACCESS LIBNAME statement, the Pass-Through Facility could be utilized to extract information from the Oracle data dictionary. The Pass-Through Facility enables a programmer to use PROC SQL to write queries which contain native Oracle SQL syntax and pass the queries directly to Oracle for processing. Both methods have distinct advantages and disadvantages. See Jesse (2011) for examples of using the Pass-Through Facility to extract information from the Oracle data dictionary.

FINDING METADATA IN THE DATA DICTIONARY VIEWS

Identifying General Information about a Table

The ALL_TABLES data dictionary view can be used to obtain information about all tables accessible to the user, such as the number of observations in each Oracle table and the average observation size. Some of the variables in this view that are of interest include NUM_ROWS, and AVG_ROW_LENGTH. The NUM_ROWS variable contains the number of observations in each table and the AVG_ROW_LENGTH variable contains the average length, in bytes, of an observation in each table. A SAS programmer may find this information to be helpful when setting the value of the READBUFF= data set option, for example.

Identifying Variable Names and Attributes

Metadata pertaining to specific variables in an Oracle table can be found in the ALL_TAB_COLUMNS data dictionary view. Some of the useful variables in this view include COLUMN_NAME, COLUMN_ID, DATA_TYPE, and CHAR_COL_DECL_LENGTH. The COLUMN_NAME variable contains the names of the Oracle variables. The COLUMN_ID variable contains the sequence number of each variable, which is the order in which the variables appear in the program data vector when the SAS/ACCESS views are referenced in a DATA step. The DATA_TYPE variable contains the Oracle data type of the variable, such as NUMBER, DATE, or VARCHAR2, for example. For character variables, the CHAR_COL_DECL_LENGTH variable contains the length of the variable.

One useful application of the ALL_TAB_COLUMNS data dictionary view involves finding all the Oracle tables that contain a variable with a particular name. For example, the following DATA step can be used to create a data set named **emp_tables** that contains the names of all tables in the **sch** schema which have a variable named **employee_id**.

```
data emp_tables(keep=table_name);
  set orasys.all_tab_columns(keep=table_name owner column_name);
  by table_name;
  where column_name = 'EMPLOYEE_ID' and owner='SCH';
run;
```

Identifying Integrity Constraints

Integrity constraints are rules that the database enforces to ensure the validity of the data contained in the tables. There are several different types of integrity constraints. We will specifically examine primary key constraints and referential constraints. A primary key constraint guarantees that a set of one or more non-missing key variables can be used to uniquely identify each observation in the table. There can be at most one primary key constraint defined on a table. A referential constraint links observations in one table to observations in another table through the use of key variables. For example, a variable included in one table might be linked to a primary key variable in another table. This type of variable is called a foreign key variable. Any values of a foreign key variable must match a value of the primary key variable in the referenced table. Other types of constraints can also be used to ensure a particular set of variables are non-missing, unique, or comply with a certain condition.

The ALL_CONSTRAINTS data dictionary view can be used to obtain information about integrity constraints, such as the constraint names, constraint types, and the names of the associated constraints in referenced tables (for referential constraints). Some of the important variables in this view include CONSTRAINT_NAME, CONSTRAINT_TYPE, SEARCH_CONDITION, R_CONSTRAINT_NAME, and INDEX_NAME. CONSTRAINT_NAME identifies the names of the integrity constraints and CONSTRAINT_TYPE identifies the types of integrity constraints. The CONSTRAINT_TYPE variable has a value of “P” on primary key constraints and a value of “R” on referential constraints. For primary key constraints, the INDEX_NAME variable contains the name of the index associated with the primary key variables. We will discuss indexes as a subsequent topic in this paper. For referential constraints, the R_CONSTRAINT_NAME variable contains the name of the constraint in the referenced table. For other constraints, the variable SEARCH_CONDITION contains the condition enforced by the constraint.

Due to the structure of Oracle databases, understanding the referential constraints can be especially important. Oracle databases are often structured such that the tables are normalized. Normalization means that, in order to reduce data redundancy, large tables are divided into smaller, less redundant tables and the tables are linked together using referential constraints. This structure enables a change in one table to be propagated throughout the rest of the database. While the benefits of normalization are significant, a normalized database model often means that the variables of interest to the SAS programmer might be scattered throughout many different Oracle tables. Therefore, SAS programmers must understand how the Oracle tables are linked together in order to "de-normalize" the database and find the information they need.

To properly understand and interpret the primary key and referential constraints, we need to know which variables are included in the constraints. To obtain this information, the constraint names must be cross-referenced with the ALL_CONS_COLUMNS data dictionary view, which contains information about all variables available to the user that are specified in integrity constraints. The variables in this view that we will utilize include CONSTRAINT_NAME, COLUMN_NAME, and POSITION. For any value of CONSTRAINT_NAME, the COLUMN_NAME variable can be used to identify the names of the key variables included in the constraints. When there is more than one key variable included in a constraint, the POSITION variable contains the position of the COLUMN_NAME value in the constraint.

Identifying Indexes

An index is a data structure that points to specific observations in a table based on the values of one or more key variables. In Oracle, an index is automatically created for each primary key constraint. When a table is large and a query extracts a relatively small percentage of the observations in the table, the use of an index will substantially improve the performance of the query. An index can be either simple, meaning that the index is based on the values of only a single variable, or composite, meaning that the index is based on the values of multiple variables. If a SAS programmer knows which indexes have been defined on a particular table, the programmer can use this information to improve efficiency. For example, knowing which Oracle table variables are indexed can help the programmer determine whether or not the DBKEY= data set option should be used. This option can improve performance when a small SAS data set is merged with an indexed Oracle table. Additionally, if the index name is known, the programmer can create an Oracle hint to direct Oracle to use a specific index using the ORHINTS= data set option. For example, suppose an Oracle index named **idx_purch3** has been defined on the **purchases** table based on the key variable **employee_id**. The following SAS code directs Oracle to use the index and creates a data set containing a subset of the table.

```
data employee_subset;
  set oralib.purchases(orhints='/*+ INDEX(PURCHASES, IDX_PURCH3) */');
  by employee_id;
  where employee_id in ('E00312' 'E00401' 'E00423');
run;
```

The reader is encouraged to see Chapman and Sridharma (2005) for more information about using the DBKEY= and ORHINTS= data set options.

The ALL_IND_COLUMNS data dictionary view can be used to obtain information about Oracle indexes, such as the index names and the indexed variables. Some of the important variables in this view include INDEX_NAME, COLUMN_NAME, and COLUMN_POSITION. The INDEX_NAME variable contains the names of the Oracle indexes. The COLUMN_NAME variable contains the names of key variables which are used by the index. When an index is composite, multiple values of COLUMN_NAME will exist for a single value of INDEX_NAME. For composite indexes, the value of COLUMN_POSITION is important. COLUMN_POSITION contains the position of the COLUMN_NAME value within the index.

THE %ORACLE_METADATA MACRO

Now that we know where to look for useful pieces of metadata in the Oracle data dictionary views, we can write a single macro program to extract all the metadata and produce a report. The %ORACLE_METADATA macro, whose complete code can be viewed in Appendix A, can be used to extract the Oracle table metadata and produce an HTML report. Values for the table name and Oracle connection options are passed as macro parameters. The Oracle connection options are optional keyword parameters which can be set to default values. An example of the HTML report produced by the macro is shown in Appendix B. Once stored as an autocall macro, the macro can be used to quickly find the useful Oracle table metadata that PROC CONTENTS won't show you.

CONCLUSION

SAS programmers need to understand the structure of their data. As a result, when the data they are analyzing resides in Oracle database tables, the programmers need to be equipped with tools to help them obtain table metadata. The Oracle data dictionary views can be queried to extract this metadata. An autocall macro that queries the data dictionary views and produces a report is a good way for SAS programmers to quickly identify helpful metadata that cannot be seen from the output of PROC CONTENTS.

REFERENCES

Chapman, David D. and Sridharma, Selvaratnam. 2005. "Using the ORACLE LIBNAME Engine to Reduce the Time it Takes to Extract Data from an ORACLE Database." Proceedings of the Northeast SAS Users Group Inc. 18th Annual Conference. Available at http://www.nesug.org/proceedings/nesug05/io/io8.pdf

Jesse, Carole. 2011. "Romancing Your Data: The Getting-to-Know-You Phase." Proceedings of the SAS Global Forum 2011 Conference. Available at http://support.sas.com/resources/papers/proceedings11/133-2011.pdf

Oracle. "The Oracle Database Reference, 10g Release 2 (10.2)." Available at http://download.oracle.com/docs/cd/B19306_01/server.102/b14237/toc.htm

ACKNOWLEDGMENTS

The author would like to thank his coworkers for taking the time to review the content of this paper as well as his management for their support, especially Leah Cox, Steve McGuire, Rachel Hayes, and Amy Wilson.

CONTACT INFORMATION

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

Jeremy W. Poling
B&W Y-12 L.L.C.
Phone: 865-574-4610
E-mail: jpoling@vt.edu

DISCLAIMER AND COPYRIGHT NOTICE

APPENDIX A: SAS CODE FOR THE %ORACLE_METADATA MACRO

```
/****************************************
This macro can be used to find metadata
related to an Oracle table. The macro
accepts five parameters.  The first
parameter, TABLE, denotes the name of the
Oracle table of interest.  There are four
additional keyword parameters that may be
used to denote the Oracle connection
options.  The macro queries the Oracle
data dictionary views that reside in the
SYS schema to obtain information such as
the number of observations in the table,
the size of the table, indexes,
constraints, and linked tables.  An HTML
report of the results is generated.
****************************************/

%macro oracle_metadata(table,
user=userID, password=xxxxxx,
path=dbname, owner=sch);

/* Use the %superq function to mask
special characters in the connection
options. */

%let user=%superq(user);
%let password=%superq(password);
%let path=%superq(path);

/* Initialize local macro variables that
are used in the macro. */

%local table_exist num_rows
  avg_row_length num_vars num_constraints
  exist_primary_key_constraints
  exist_referential_constraints
  primary_key_constraint
  primary_key_index constraint_names
  r_constraint_names names_of_constraints
  num_indexes indexed
  foreign_key_constraints
  exist_foreign_key_tables;

/* Use a LIBNAME statement to connect to
the Oracle data dictionary views. */

libname orasys oracle user="&user"
  password="&password"  path="&path"
  schema='sys' defer=no readbuff=1000;

/* If the ORASYS library still does not
exist, issue an error message and
terminate execution of the macro. */

%if %sysfunc (libref(orasys)) ne 0 %then
%do;
  %put ERROR:  Unable to connect to the
Oracle data dictionary.;
  %return;
%end;

/* Convert the Oracle table name and
owner to upper case. */

%let table=%qupcase(%superq(table));
%let owner=%qupcase(%superq(owner));

/* Find the number of observations and
the observation length. This information
is stored in macro variables. If no
records are found in the ALL_TABLES view,
then create a macro variable to indicate
that the specified table does not exist.
*/

data _null_;
  if _n_=1 and eof then call symputx
   ("table_exist", "No");
  set orasys.all_tables(keep=table_name
    owner num_rows avg_row_len) end=eof;
  where table_name="&table" and
    owner="&owner";
  call symputx("num_rows", num_rows);
  call symputx("avg_row_length",
    avg_row_len);
run;

/* If the specified table does not exist
then issue an error message to the log
and terminate macro execution. */

%if &table_exist=No %then %do;
  %put ERROR:  Oracle table "&table" does
not exist.;
```

```
  %return;
%end;

/* Create a data set containing variable
information for the table of interest.
Also, create a macro variable to denote
the total number of variables in the
Oracle table. */

data table_vars(drop=table_name owner
  num_vars);
  if _n_=1 and eof then do;
    call symputx("num_vars", "0");
    stop;
  end;
  set orasys.all_tab_columns
    (keep=table_name owner column_id
    column_name data_type
    char_col_decl_length) end=eof;
  by column_name;
  where table_name="&table" and
    owner="&owner";
  num_vars+1;
  if eof then call symputx("num_vars",
    num_vars);
run;

/* Two data sets are created which
contain constraint data, one for
referential constraints and one for other
constraints.  Macro variables are created
to indicate the total number of
constraints and the existence of primary
key and referential constraints.  If the
table has a primary key constraint, then
two more macro variables are created
which will resolve to the constraint name
and the index name of the primary key.
Additionally, when referential
constraints exist, two macro variables
are created which will resolve to the
names of all referential constraints and
the names of the associated constraints
in the referenced tables.*/

data other_constraints
  (keep=constraint_name condition)
  referential_constraints
  (keep=constraint_name
    r_constraint_name);
  if _n_=1 and eof then do;
    call symputx("num_constraints", "0");
    call symputx
      ("exist_primary_key_constraints",
        "No");
    call symputx
      ("exist_referential_constraints",
        "No");
    stop;
  end;
  set orasys.all_constraints
    (keep=table_name owner
      constraint_name constraint_type
      search_condition r_constraint_name
      index_name
    rename=(constraint_type=type
      search_condition=condition))
    end=eof;
  where table_name="&table" and
    owner="&owner";
  length constraint_names
    r_constraint_names $ 2000
    exist_primary_key_constraints
    exist_referential_constraints $ 3;
  retain constraint_names
    r_constraint_names
    exist_primary_key_constraints 'No'
    exist_referential_constraints 'No';
  condition=left(condition);
  if type='P' then do;
    exist_primary_key_constraints='Yes';
    call symputx
      ("primary_key_constraint",
        constraint_name);
    call symputx("primary_key_index",
      index_name);
  end;
  else if type='R' then do;
    exist_referential_constraints='Yes';
    output referential_constraints;
    constraint_names=catx(' ',
      constraint_names,
      strip(constraint_name));
    r_constraint_names=catx(' ',
      r_constraint_names,
      strip(r_constraint_name));
  end;
  else output other_constraints;
  num_constraints+1;
  if eof then do;
    call symputx("num_constraints",
      num_constraints);
    call symputx
      ("exist_primary_key_constraints",
      exist_primary_key_constraints);
    call symputx
      ("exist_referential_constraints",
      exist_referential_constraints);
    call symputx("constraint_names",
      constraint_names);
    call symputx("r_constraint_names",
      r_constraint_names);
  end;
run;

/* Create a view that will contain any
table names and variable names involved
in a set of constraints.  The set of
constraints used is based on the value of
a macro variable named
NAMES_OF_CONSTRAINTS.  This macro
variable is resolved at execution time
using the SYMGET function.  Instead of
having a separate observation for each
variable involved in each constraint, the
view contains one observation for each
constraint and the variable names are
concatenated according to their position
within the constraint.  */

data constraint_variable_names
  (keep=constraint_name table_name
  variables)
  /view=constraint_variable_names;
  set orasys.all_cons_columns
    (keep=table_name owner
    constraint_name column_name
    position);
  by constraint_name position;
  where owner="&owner" and
    findw(symget('names_of_constraints'),
    strip(constraint_name));
  length variables $ 2000;
  retain variables;
  if first.constraint_name then call
    missing(variables);
  variables=catx(', ', variables,
    strip(column_name));
  if last.constraint_name then output;
run;

/* If referential constraints exist on
the Oracle table, then additional
information is extracted from the data
dictionary, including the names of the
referenced tables and the constrained
variable names. */

%if &exist_referential_constraints=Yes
%then %do;

/* Use the CONSTRAINT_VARIABLE_NAMES view
to add a variable containing the
constrained variable names to the
REFERENTIAL_CONSTRAINTS data set. */

  proc sort data=referential_constraints;
    by constraint_name;
  run;

  %let names_of_constraints=
    &constraint_names;

  data referential_constraints
    (drop=table_name);
    merge referential_constraints
          constraint_variable_names;
    by constraint_name;
  run;

/* Use the CONSTRAINT_VARIABLE_NAMES view
again to add variables to the
REFERENTIAL_CONSTRAINTS data set which
denote the referenced tables and the
names of the constrained variables in the
referenced tables. */

  proc sort data=referential_constraints;
    by r_constraint_name;
  run;

  %let names_of_constraints=
    &r_constraint_names;

  data referential_constraints;
    merge referential_constraints
      constraint_variable_names
      (rename=(constraint_name=
        r_constraint_name
        table_name=r_table_name
        variables=r_variables));
    by r_constraint_name;
  run;

/* Sort the data set for subsequent
printing. */

  proc sort data=referential_constraints;
    by constraint_name;
  run;

%end;

/* Determine the number of Oracle indexes
that exist on the given table and store
this information in a macro variable.  A
data set is created that contains the
name of each index and the variables that
comprise the index. For composite indexes
that involve multiple variables, the
variable names are concatenated into a
single variable so that the output data
set contains only one observation per
index name. */
```

```
data indexes(keep=index_name variables
  primary_key);
  if _n_=1 and eof then do;
    call symputx("num_indexes", "0");
    stop;
  end;
  set orasys.all_ind_columns
    (keep=table_name table_owner
      index_name column_name
      column_position)
    end=eof;
  by index_name column_position;
  where table_name="&table" and
    table_owner="&owner";
  length variables $ 2000
    primary_key $ 3;
  retain variables primary_key;
  if first.index_name then do;
    call missing(variables, primary_key);
    num_indexes+1;
  end;
  if index_name="&primary_key_index" then
    primary_key='Yes';
  variables = catx(', ', variables,
    strip(column_name));
  if last.index_name then output;
  if eof then call symputx("num_indexes",
    num_indexes);
run;

/* Create a Yes/No macro variable to
indicate whether or not the Oracle
table is indexed. */

%let indexed=%sysfunc(ifc(&num_indexes>
  0, Yes, No));

/* If the table has a primary key
constraint, then create a data set
containing the names of all other Oracle
tables with a foreign key which is the
primary key in the selected table. A
macro variable is created to indicate
whether there exist any such tables.  If
there are tables with a foreign key which
is the primary key in the selected table,
then another macro variable is created
which will resolve to the constraint
names in the referenced tables. */

%if &exist_primary_key_constraints=Yes
%then %do;

  data foreign_key_tables(keep=table_name
    constraint_name);
    set orasys.all_constraints
      (keep=table_name owner
      constraint_name r_constraint_name)
    end=eof;
    by constraint_name;
    where owner="&owner" and
      r_constraint_name=
       "&primary_key_constraint";
    length foreign_key_constraints $
      2000;
    retain foreign_key_constraints;
    foreign_key_constraints=catx(' ',
      foreign_key_constraints,
      strip(constraint_name));
    if eof then do;
      call symputx
        ("foreign_key_constraints",
        foreign_key_constraints);
      call symputx
        ("exist_foreign_key_tables",
        "Yes");
    end;
  run;

/* If there are tables with a foreign key
which references the primary key
constraint in the selected table, then
use the CONSTRAINT_VARIABLE_NAMES view to
find the names of the constrained
variables in the these tables.  */

  %if &exist_foreign_key_tables=Yes
    %then %do;

    %let names_of_constraints=
      &foreign_key_constraints;

    data foreign_key_tables;
      merge foreign_key_tables
        constraint_variable_names;
      by constraint_name;
    run;

/* Sort the data set for subsequent
printing. */

    proc sort data=foreign_key_tables;
      by table_name;
    run;

  %end;
%end;

/* Use the macro variables previously
created to construct a data set with
table attributes which will be printed in
the report */
```

```
data table_attributes;
  length attribute $ 26 value $12;
  attribute='Number of Observations';
  value="&num_rows"; output;
  attribute='Number of Variables';
  value="&num_vars"; output;
  attribute='Average Observation Length';
  value="&avg_row_length"; output;
  attribute='Indexed'; value="&indexed";
  output; attribute='Number of Indexes';
  value="&num_indexes"; output;
  attribute='Number of Constraints';
  value="&num_constraints"; output;
run;

/* Produce the Oracle table metadata HTML
report using a sequence of PROC PRINT
steps. */

ods listing close;
ods html
  file="Oracle_table_metadata.html"
  style=minimal;

title "Summary of Metadata for the &table
Table";
title3 "Table Attributes";
footnote;
proc print data=table_attributes noobs
  label;
  label attribute="Attribute"
  value="Value";
run;

title "Alphabetic Listing of Table
Variables";
proc print data=table_vars label;
  label column_id='#'
    Column_Name="Variable Name"
    data_type="Type"
    char_col_decl_length="Char Length";
  var column_id column_name data_type
    char_col_decl_length;
run;

title "General Table Constraints";
proc print data=other_constraints
  label;
  label
    constraint_name='Constraint Name'
    condition='Condition';
  var constraint_name condition;
run;

%if &exist_referential_constraints=Yes
%then %do;

  title "Referential Table Constraints";
  proc print data=referential_constraints
    label;
    label
      constraint_name='Constraint Name'
      variables="Constrained Variables in
the &table Table"
      r_constraint_name="Referential
Constraint Name"
      r_table_name="Referenced Table"
      r_variables='Referenced Table
Variables';
    var constraint_name variables
      r_constraint_name r_table_name
      r_variables;
  run;

%end;

title "Indexes";
proc print data=indexes label;
  label index_name='Index Name'
    primary_key='Primary Key'
    variables='Variables';
  var index_name primary_key variables;
run;

%if &exist_foreign_key_tables=Yes %then
  %do;

  title "Tables Where a Foreign Key
References the Primary Key Constraint,
%qsysfunc(strip(&primary_key_constraint))
";
  proc print data=foreign_key_tables
    label;
    label table_name='Table Name'
      constraint_name='Constraint Name'
      variables='Variables';
    var table_name constraint_name
      variables;
  run;

%end;

ods html close;
ods listing;

%mend;
```

APPENDIX B: SAMPLE OF HTML OUTPUT PRODUCED BY THE %ORACLE_METADATA MACRO

Summary of Metadata for the PURCHASES Table

Table Attributes

Attribute	Value
Number of Observations	184279
Number of Variables	7
Average Observation Length	50
Indexed	Yes
Number of Indexes	4
Number of Constraints	5

Alphabetic Listing of Table Variables

Obs	#	Variable Name	Type	Char Length
1	4	CUSTOMER_ID	VARCHAR2	12
2	5	DATE_PURCHASE	DATE	.
3	3	EMPLOYEE_ID	VARCHAR2	12
4	2	PURCHASE_CODE	VARCHAR2	1
5	7	PURCHASE_COMMENTS	VARCHAR2	1000
6	1	PURCHASE_ORDER_ID	VARCHAR2	12
7	6	TOT_PAYMENT_DUE_AMOUNT	NUMBER	.

General Table Constraints

Obs	Constraint Name	Condition
1	SYS_C000122	"PURCHASE_ORDER_ID" IS NOT NULL
2	CONS_PURCH4	PURCHASE_CODE IN ('O', 'P', 'S')

Referential Table Constraints

Obs	Constraint Name	Constrained Variables in the PURCHASES Table	Referential Constraint Name	Referenced Table	Referenced Table Variables
1	CONS_PURCH2	CUSTOMER_ID	CONS_CUST1	CUSTOMER	CUSTOMER_ID
2	CONS_PURCH3	EMPLOYEE_ID	CONS_EMP1	EMPLOYEE	EMPLOYEE_ID

Indexes

Obs	Index Name	Primary Key	Variables
1	IDX_PURCH1	Yes	PURCHASE_ORDER_ID
2	IDX_PURCH2		DATE_PURCHASE
3	IDX_PURCH3		EMPLOYEE_ID, CUSTOMER_ID
4	IDX_PURCH4		CUSTOMER_ID

Tables Where a Foreign Key References the Primary Key Constraint, CONS_PURCH1

Obs	Table Name	Constraint Name	Variables
1	PAYMENTS	CONS_PAY2	PURCHASE_ORDER_ID
2	PURCHASE_PRODUCTS	CONS_PURCHPROD2	PURCHASE_ORDER_ID