Optimizing Data Extraction from Oracle® Tables

Caroline Bahler, Meridian Software, Inc.

Abstract

The SAS/ACCESS® product for Oracle® offers a fast and easy interface for reading Oracle tables - access views or SQL Pass-Thru facility. However, if your extraction requirements include joining Oracle tables or extracting date ranges then you need to know Oracle's "flavor" of SQL. In addition, efficient use of SAS/ACCESS methods may require some knowledge of how the Oracle database was designed and what Oracle operational options are in effect. The objective of this paper is to discuss some of the potential pitfalls and efficiencies when working with SAS/ACCESS to extract information from Oracle. This paper assumes the reader is familiar with the SAS/ ACCESS product and specifically targets SAS/ACCESS and Oracle operational and SQL options that can be used to optimize data retrieval.

Extracting Data from Oracle

ACCESS Views

An Access view creates a “road map” to the location of a table and permits data extraction to occur. Access views are created using PROC ACCESS. A typical invocation of the ACCESS procedure would be:

```
proc access dbms=oracle;

/* create access descriptor */
 create work.sales.access;
user=MyUserid;
   orapw=MyPassword; path='@ORAPATH.WORLD';
table=acme_na.Sales_NorthAmer_1999;
assign=yes;
list all;

/* create view descriptor */
 create work.sales.view;
select all;

run;
```

(Note - @ is an Oracle specific term used to designate the correct Oracle SQLNET® path. Whether the @ is utilized is dependent upon how Oracle is set-up at your site.)

In the example above a view is created instead of a data set. The advantage of this is that the view only needs to be created once and then can be reused. A new view needs to be generated only when the Oracle table is modified (new columns added or new indexes NOT new rows). Therefore, a permanent set of views can be created that can be used over and over without worrying about new rows within the table.

Tip 1 – Assess when to allow SAS to automatically assign variable name to Oracle table columns.

In the example above, the use of the ASSIGN statement automatically creates a SAS variable name for each Oracle column. The new variable name consists of the first eight (8) characters of each column name[1]. The problem with allowing SAS to automatically assign variable name is illustrated below:

Oracle Column Name	SAS Variable Name
Product_Number	Product
Product_Type	Product1
Product_Serial_Number	Product2
Product_Sales	Product3

In each case the first eight (8) characters is product_. SAS® handles this situation by adding a number to the end of each variable name. Unique variable names are created, but the names are not indicative of the data stored within each variable. To assign unique variable names to these columns, use the RENAME statement within the CREATE statement to individually assign variable names to columns. So from the previous example:

```
proc access dbms=oracle;
/* create access descriptor */
 create work.sales.access;
  user=MyUserid;
  orapw=MyPassword;
  table=acme_na.Sales_NorthAmer_1999;
  path='@ORAPATH.WORLD';
  assign=yes;
  /* rename oracle columns to meaningful SAS */
  /* variable names                                          */
  rename  Product_Number = prodno
          Product_type = prodtype
          Product_Serial_Number=serialno
          Product_Sales = prodsale;

  list all;

/* create view descriptor */
 create work.sales.view;
   select all;
run;
```

Note: The RENAME statement must be used when the access descriptor is created. RENAME and ASSIGN=YES are mutually exclusive and cannot be used together.

Therefore, if the column names are eight (8) characters or less or uniquely named, then having SAS assign the name is appropriate.

Tip 2 – Efficiency considerations

The amount of data within a table will directly affect the time and resources needed in both SAS and Oracle to extract the information requested by the view. An ACCESS view reads and extracts from the Oracle table every time it is utilized. This is not a problem if the table from which you are extracting is a small lookup table containing a few hundred rows. However, if the table contains many thousands of rows, it may be more efficient to utilize the view to create a data set[1]. This data set is then available for use. The key factors that should be evaluated when considering this strategy are:

1. How many times is the table accessed within a program or application? If the table is accessed several times within a program or application, then creating a data set is a good choice.
2. How much temporary space is available for SAS data?

3. Can the data extracted from the table be subset when the table is accessed by using a WHERE statement? Use of an efficient where clause can reduce the amount of data and the amount of time needed for data extraction.

4. Can you reduce the number of columns you are selecting? SAS must convert the data within each column selected from Oracle to SAS format. So, by decreasing the number of columns selected, the resources and time needed to extract data is decreased. You can select columns when a view is created by using a KEEP or DROP statement within a DATA step or a VAR statement within a procedure.

   You may need to benchmark to determine whether creation of a data set or a view is the more efficient option.

   For example:

```
proc access dbms=oracle;
/* create access descriptor */
 create work.sales.access;
  user=MyUserid;
  orapw=MyPassword;
  table=acme_na.Sales_NorthAmer_1999;
  path='@ORAPATH.WORLD';
  assign=yes;
  /* rename oracle columns to meaningful SAS */
  /* variable names                                          */
  rename  Product_Number = prodno
          Product_Type = prodtype
          Product_Serial_Number=serialno
          Product_Sales = prodsale;

  list all;

/* create data set  */
 create work.sales;
  selectProduct_Number Product_Type
       Orderno Product_Sales;
   subset where Product_Sales > 100000;
run;
```

   Note: The where clause of the SUBSET statement is sent "as is" to Oracle, so it must contain correct Oracle syntax.

When to Use Access Views

- Information from only one table at a time is required.
- You are extracting data from a table with a small number of rows.
- You are accessing table information one row at a time or allowing a user to update data within a table row.

SQL Pass-Thru

PROC SQL allows you to execute Oracle[1] commands or create SAS data sets/views from Oracle queries through the SQL Pass-Thru facility.

Connection options - SQL Pass-Thru requires that a connection to Oracle first is established before an Oracle command or query can be executed. The CONNECT statement requires the same information as in PROC

1. All Oracle® comments refer to Oracle version 7, 7.1, 7.2

ACCESS user id, password, Oracle path. For example the following statements will connect PROC SQL to Oracle with the same parameters specified for PROC ACCESS.

```
proc sql;
  Connect to oracle(
userid='MyUserid'
orapw='MyPassword'
       path="@ORAPATH.WORLD");

       ...
       quit;
```

Tip 3 – Other CONNECT options. SQL Pass-Thru has two (2) other connection options, buffsize and preserve_comments, that can decrease the amount of time required to extract data from the results of an Oracle query.

BUFFSIZE – this option specifies the number of rows that will be transferred from Oracle to SAS when a fetch occurs, i.e. the number of rows that go into a buffer which moves the Oracle table rows from Oracle to SAS[1]. The default BUFFSIZE is 25 rows and the maximum is 32,767 rows. Increasing the number of rows within the buffer can enhance extraction performance.

Depending upon your particular hardware and Oracle setup using a BUFFSIZE between 5,000 and 10,000 will give the best performance in most cases. Somerville and Cooper in their SUGI 23 paper ran tests with Oracle tables containing 5.8 million records and found that a BUFFSIZE of 5000 gave them their best results[3]. Note that BUFFSIZE can not be set within PROC ACCESS or DBLOAD.

An example of using BUFFSIZE:

```
proc sql;
  Connect to oracle(
userid='MyUserid'
orapw='MyPassword'
       path="@ORAPATH.WORLD,buffsize=5000");

       create table sales as
         select *
            from connection to Oracle
             (select *
                 from acme_na.Sales_NorthAmer_1999);
       quit;
```

When to Use SQL Pass-Thru

[2] The output data set requires information from more than one Oracle table.

[2] Subqueries – this is a technique that uses the information from one table to subset another. Subquerying can be a very efficient way of selecting only the information needed from a table. For example –

```
   proc sql;
    connect to oracle(userid='MyUserid'
orapw='MyPassword'
       path="@ORAPATH.WORLD,
       buffsize=5000");
```

```
create table sales as
 select *
   from connection to Oracle
    (select *
        from acme_na.Sales_NorthAmer_1999
      where CustomerID in
         (select CustomerID
              from acme_na.Customer
  where region like 'NORTH%')
       ) ;
quit;
```

In the example above, a subquery made sense because we wanted just the customers residing in the northern regions. It is also efficient since only the required information was selected from both tables.

- Joins – Two or more tables are joined using either an inner or outer join. Note: It is a good idea to evaluate whether it is more efficient to join the tables within Oracle or SAS.
- Extracting data from a table with a large number of rows. The use of the BUFFSIZE option within the path connect parameter greatly speeds up extracting data from a large table.
- Need to perform an Oracle command. There are instances where the ability to submit an Oracle command first is important. For example you need to load a table from a SAS data set using PROC DBLOAD, but first you need to drop the table so that new columns can be added.

Querying the Oracle Data

Oracle Meta Data

Oracle metadata consists of an extensive set of tables containing information about all database tables, user privileges, etc. The main meta data tables of interest are those containing information about the tables you need to access to obtain the information needed. Note: While the term table is used for descriptive purposes, Oracle considers all data dictionary tables to be views. Table 1 lists all Oracle dictionary tables[4] of interest.

Tip 4 – Listing Oracle table columns. A quick way to get basic information about the columns in an Oracle table is to use the DESC command (it does not need to be capitalized) in SQLPLUS®. The DESC command prints a listing of all columns in alphabetical order with their associated format and length.

Why do you need metadata?

In many cases, a programmer can go to the dba and get a printed copy of the data dictionary including the entity relationship diagrams. However that may not always be the case, so the programmer is forced to dig for the information that is needed (i.e. the dba for table names and meta data for table information). Also remember Oracle is a relational database and a single database can contain hundreds (even thousands) of tables! Relationships (primary and foreign keys) between tables become extremely important when joining these tables. Finally, meta data can be used to write queries on the "fly" by utilizing the data dictionary within query–writing macros or SCL.

Table 1. Oracle Data Dictionary Tables[4].

Table Name*	Description

USER_ : these are views containing information about tables owned by a particular user id

USER_TABLE	Table listing
USER_TAB_COLUMNS	Column listing by table
USER_CONSTRAINTS	Listing of constraints defined for a table (primary, foreign, unique).
USER_CONS_CONSTRAINTS	Used with USER_CONSTRAINTS to define a table's primary and foreign keys.
USER_INDEXES	Listing of indexes defined for a table.
USER_IND_COLUMNS	Used with USER_INDEXES to defined which columns within a table are used within an index. Note – Oracle automatically creates an unique index for all unique and primary keys.

ALL_ : these are views containing information about all tables accessible by a particular user id

ALL_TABLE	Table listing
ALL_TAB_COLUMNS	Column listing by table
ALL_CONSTRAINTS	Listing of constraints defined for a table (primary, foreign, unique).
ALL_CONS_CONSTRAINTS	Used with ALL_CONSTRAINTS to define a table's primary and foreign keys.
ALL_INDEXES	Listing of indexes defined for a table.
ALL_IND_COLUMNS	Used with ALL_INDEXES to defined which columns within a table are used within an index. Note – Oracle automatically creates an unique index for all unique and primary keys.

*Note: In addition, Oracle is case sensitive when using these table(view) names in queries the table name MUST be capitalized.

Where to do joins

In general, Oracle tables should be joined within Oracle, because all optimization features are available to enhance the performance of a join. To evaluate where to join Oracle tables, you should ask the following questions:

² What type of join is needed to provide the required information in the resulting data set?

Determining what the result data set should look like will identify the type of join needed. Oracle has two types of joins available – an inner and an outer join[5,6].

§ The inner join produces a result data set that contains only those rows that match exactly in all parent tables.

§ The outer join will produce a result data set that contains all rows from all parent tables (similar to using a MERGE statement with a BY statement in a DATA step)[5].

Make sure that using either type of SQL join will provide the needed results. Remember, within SQL (SAS and Oracle) both types of joins can result in Cartesian products, so check the parent tables carefully to prevent unintended extra rows[7].

However, there are cases where Oracle does not have the tools to allow for the type of join necessary. Table 2 lists all of the types of joins available within SAS and Oracle. Oracle can not perform joins B or D but SAS can[8].

To perform an inner join within Oracle, use the following syntax:

```
create table sales as
  select *
     from connection to Oracle
      (select b.CompanyName, a.Sales
          from acme_na.Sales_NorthAmer_1999 a,
                  acme_na.Customer b
        where a.CustomerId=b.CustomerId);
```

To perform an outer join, use the following syntax:

```
create table sales as
  select *
     from connection to Oracle
      (select a.Region, a.Territory, b.Sales_Rep
          from acme_na.NorthAmer_Region a,
                  acme_na.Sale_Representative b
        where a.Region=b.Region(+));
```

² Can the table be joined by primary or foreign keys? Are indexes available?

Using the primary and foreign keys within tables improves the performance of joins. This is because Oracle automatically creates indexes on the tables for these keys and the optimizer within Oracle will use these indexes during any join[4]. In addition, all other indexes on a table are available so performance of the join will be enhanced. By moving the tables into SAS data sets, these indexes

Table 2. Types of joins and their availability within SAS and Oracle.

		Availability	
Result Data Set	**Type of Join**	**SAS**	**Oracle**
A. All data values from all parent* tables.	Full Outer Join	**X**	**X**
B. All data values from a single parent table (base) and all data values from the other table(s) that match the data values of the joining variables within the base.	Non-base parent data set(s) are used as ² "Look-up" table(s) ² Right or Left outer join.	**X**	
C. Only those data values from all parent tables that contain the same data values within the joining variables.	Inner Join	**X**	**X**
D. Placement of parent tables side by side	One-to-one Merge	**X**	

E. Expansion of the result data set to include all levels of a non-common variable (Cartesian products).	Many-to-many Join (Inner or Outer Join)	**X**	**X**

* Parent table = one of the tables joined to produce the result data set.

are lost and the join is performed on unindexed variables.

² What are the sizes of the tables to be joined?

The decision to join a set of tables within Oracle or SAS can be affected by table size. To join the tables within SAS, all tables must first be extracted to SAS and then joined. The larger the table, the more time it takes to extract it to SAS. CPU time is expended before the join takes place. In comparison, by joining the tables within Oracle with indexes in place, the join takes less time and only one extract needs to take place.

However, there is a circumstance where joining the tables within SAS may have to occur – when Oracle does not have enough temporary tablespace (work space) for the joins to occur. Finally, if one or more of the tables is extremely large, (1 million plus records) then the join may have to occur in smaller subsets with the total result data set being concatenated within SAS.

Is the database optimized for OLTP (on-line transaction processing) or as a data warehouse?

A database that is optimized for OLTP will be extremely fast at getting all of the information required about a single customer for example. Other types of joins may be much slower within that type of database than within SAS. In addition, your DBA may not approve of any large CPU drain during “peak” usage times. In contrast, a database optimized for data warehouse usage will be designed to optimize joins requiring large amounts of data.

² What does your DBA say?

Finally, the DBA is your best source of information about the database. They can suggest “best” practices for joining tables within the database.

Oracle SQL Tips

Dates

Oracle Date Formats[9]

MM	Number of month for example 12
MON	Three letter abbrev for example NOV
MONTH	Full month name
DD	Day of month
YYYY	Four digit year
YY	Two digit year
HH	Hour 1-12
HH24	Hour 1-24
MI	Minutes
SS	Seconds

Some useful Oracle date functions[9] are

- SysDate – this is equivalent to the DATE() or TODAY() functions within SAS and returns the current date and time.
- TO_CHAR(*date*,'*format*') – used in queries to change an Oracle datetime value into a formatted value. Similar in function to using the following SAS syntax - put(*date*,*mmddyy*10.). This format can be used within a query as follows:

```
create table sales as
 select *
    from connection to Oracle
     (select b.CompanyName, a.Sales
         from acme_na.Sales_NorthAmer_1999 a,
                 acme_na.Customer b
       where a.CustomerId=b.CustomerId
           and to_char(Updt_date,'MM/DD/YYYY') =
                   '01/01/2000');
```

  Using TO_CHAR can be tricky since you are specifically turning a numeric datetime value into a character. You need to be careful how you use this function, in general TO_CHAR is more useful in an equality situation.

- TO_DATE('*date*','*format*') – from my own experience and talking to knowledgeable Oracle programmers this is the preferred function. For example:

```
create table sales as
  select *
     from connection to Oracle
      (select b.CompanyName, a.Sales
          from acme_na.Sales_NorthAmer_1999 a,
                  acme_na.Customer b
        where a.CustomerId=b.CustomerId
            and Updt_date between
             to_date('01/01/2000','MM/DD/YYYY') and
             to_date('02/15/2000','MM/DD/YYYY')  );
```

  Tip 5 – Feeding dates into an Oracle query.

Using the same query as above, we can generalize for dates as follows:

```
Data _null_;
 To=today();
 From=intx('week',to,-8);

 Call symput('from',"'"||put(from,mmddyy10.)||"'"");
 Call symput('to',"'"||put(to,mmddyy10.)||""'"");
Run;

Proc sql;
 Connect to oracle(
userid='MyUserid'
```

```
orapw='MyPassword'
        path="@ORAPATH.WORLD,buffsize=5000");

        create table sales as
          select *
             from connection to Oracle
              (select b.CompanyName, a.Sales
                  from acme_na.Sales_NorthAmer_1999 a,
                          acme_na.Customer b
                where a.CustomerId=b.CustomerId
                    and Updt_date between
                     to_date(&from , 'MM/DD/YYYY') and
                     to_date(&to , 'MM/DD/YYYY')  );


  disconnect from oracle;

quit;
```

Note: Double quotes (“) are not used around the macro variables as in SAS. Instead, the quotes are included as part of the macro variable value. Macro variables are extremely useful within SQL Pass-Thru in automating the Oracle query.

One use of macro variables and meta data tables is to determine the fields within a table. The resulting data set can be used to build a query by passing the fields into the query as macro variables.

Other Tricks of the Trade

Ø Quotes – the use of quotes can be tricky within Oracle. As stated above, it is best to pass the quotes within the macro variable value. Oracle does not handle double quotes (“) within the WHERE portion of a query. Use of double quotes results in a cryptic error message. Instead, incorporate the quotes into the macro variable passed.

In the example below, each of the passed macro variables contains the necessary quote for the character values.

```
        Data _null_;
         Set param;
         To=today();
         From=intx('week',to,-8);

         Call symput('state',"'"||trim(state)||"'");
         Call symput('from',"'"||put(from,mmddyy10.)||"'"):
         Call symput('to',"'"||put(to,mmddyy10.)||"'");
        Run;

        Proc sql;
 Connect to oracle(
userid='MyUserid'
orapw='MyPassword'
        path="@ORAPATH.WORLD,buffsize=5000");

        create table sales as
          select *
             from connection to Oracle
```

```
            (select b.CompanyName, a.Sales
                from acme_na.Sales_NorthAmer_1999 a,
                        acme_na.Customer b
                where a.CustomerId=b.CustomerId
                    and Updt_date between
                      to_date(&from , 'MM/DD/YYYY') and
                      to_date(&to , 'MM/DD/YYYY')
                    and state = &state );


  disconnect from oracle;

quit;
```

Tip 6 - Handling a quote embedded within a field value. If a field value has an embedded quote in its value, a comment field or company name for example, then you should use special care when you are within a query. In the following example a title contains a single quote:

Mr. Roger's Neighborhood

To use this title in a PROC SQL query, surrounding the title with double quotes is sufficient. In an Oracle query, the following is needed:

'Mr. Roger''s Neighborhood'

Note: the two single quotes are side by side[5]. Therefore, any text value that may contain quotes could be handled as follows:

```
Data _null_;
 Set titles end=eof;
   Where title contains "SESUG";
Quottest=index(title,"'");
If quottest > 0 then
 Title=scan(title,1,"'")||"''"||scan(title,2,"'");

If _n_=1 then titles="'"||trim(title)||"'";
 Else title=trim(title)||"',"|| trim(title)||"'";

If eof then call symput('titles'title');
Run;

Proc sql;
 Connect to oracle (
userid='MyUserid'
orapw='MyPassword'
       path="@ORAPATH.WORLD,buffsize=5000");

 create table proceeds as
   select *
      from connection to oracle
       (select author_lastname lastname,
                  author_firstname firstnme,
                   title
          from bibliography
```

```
         where title in ( &titles ) );

    disconnect from oracle;
quit;
```

Ø “Temporary Tables” – within Oracle, each user has personal tablespace for creating tables. This tablespace can be put to use for creating temporary query result tables depending upon their size. The idea here is to run a query subsetting a much larger table and then use the results of that query to subset other tables. For example –

```
Data _null_;
 Set titles end=eof;
   Where title contains “SESUG”;
Quottest=index(title,”’”);
If quottest > 0 then
 Title=scan(title,1,”’”)||”’’”||scan(title,2,”’”);

If _n_=1 then titles=”’”||trim(title)||”’”;
 Else title=trim(title)||”’,”|| trim(title)||”’”;

If eof then call symput(‘titles’title’);
Run;

Proc sql;
 Connect to oracle (
userid=’MyUserid‘
orapw=’MyPassword’
       path=”@ORAPATH.WORLD,buffsize=5000”);

    execute
       (create table temp as
          select author_lastname lastname,
                 author_firstname firstnme,
                  title
           from bibliography
          where title in ( &titles ) )
      by oracle;

     create table advtut as
      select *
         from connection to oracle
        (select *
           from temp
         where title like ‘SQL%’ );

    execute (drop table temp) by oracle;

    disconnect from oracle;
quit;
```

Note: The temporary table needs to be dropped when you are finished! A permanent Oracle table is being created, so this house-cleaning step is important.

What's New - SAS 7 and Beyond

Versions 7 and 8 have several new features[10]:

- Dynamic DBMS Engines – this essentially replaces SAS/ACCESS views and descriptors! In version 7 the LIBNAME statement now connects to the DBMS server. For example:

```
Libname oralib oracle user=MyUserId password=MyPassword path=@ORACLE.WORLD;
```

  This means that all Oracle tables to which you have access are now available to be used within your program.

- Version 7 supports long variable names so truncation of Oracle column names is no longer a problem.

References

1. SAS Institute Inc. 1993. SAS/ACCESS Interface to Oracle Usage and Reference. Version 6, Second Edition, Cary, NC. SAS Institute Inc.
2. SAS Institute Inc. 1994. SAS/ACCESS Software for Relational Databases: Reference, Version 6, First Edition, Cary, NC. SAS Institute Inc.
3. Somerville, Clare and Copper, Clive. 1998. Optimizing SAS® Access to an Oracle Database in a Large Data Warehouse. Proceedings of the Twenty-Third Annual SAS Users Group International Conference, Cary, NC. SAS Institute. pp 511-520.
4. Koch, George and Loney, Kevin. 1995. Chapter 24. The Hitchhiker's Guide to the ORACLE7 Data Dictionary. Oracle: Complete Reference Third Edition. Oracle Press™, Berkeley, CA. Osborne McGraw-Hill Inc. pp 540-586
5. Koch, George and Loney, Kevin. 1995. Chapter 25. Alphabetical Reference. Oracle: Complete Reference Third Edition. Oracle Press™, Berkeley, CA. Osborne McGraw-Hill Inc.
6. Koch, George and Loney, Kevin. 1995. Chapter 10. When One Query Depends Upon Another. Oracle: Complete Reference Third Edition. Oracle Press™, Berkeley, CA. Osborne McGraw-Hill Inc.
7. Koch, George and Loney, Kevin. 1995. Chapter 9. Grouping Things Together. Oracle: Complete Reference Third Edition. Oracle Press™, Berkeley, CA. Osborne McGraw-Hill Inc.
8. Bahler, Caroline. 1996. It Takes at Least Two to Tango -A Data Set Joining Primer. Proceedings of the Twenty-Second Annual SAS Users Group International Conference, Cary, NC. SAS Institute. pp 190-198.
9. Koch, George and Loney, Kevin. 1995. Chapter 7. Dates: Then, Now, and the Difference. Oracle: Complete Reference Third Edition. Oracle Press™, Berkeley, CA. Osborne McGraw-Hill Inc. pp. 173-189.
10. Gona, Vino and Van Wyk, Jana. 1998. Version 7 Enhancements to SAS/ACCESS Software. Proceedings of the Twenty-Third Annual SAS Users Group International Conference, Cary, NC. SAS Institute. pp 336-341.

Trademarks

Contact Information

Caroline Bahler

Meridian Software, Inc.

12204 Old Creedmoor Road

Raleigh, NC 27613

(919) 518-1070

merccb@meridian-software.com