CHAPTER 13

Using DBMS Data with the SQL Pass-Through Facility

Introduction

This topic presents examples of accessing and updating DBMS data through the SQL Procedure Pass-Through Facility.

Note: It is recommended that you use the new SAS/ACCESS LIBNAME statement to access your DBMS data more easily and directly and to take full advantage of Version 7 enhancements. See “SAS/ACCESS LIBNAME Statement” on page 24 for more information about the new LIBNAME statement.

Retrieving DBMS Data with a Pass-Through Query

This section describes how to retrieve DBMS data by using the statements and components of the SQL Procedure Pass-Through Facility to access DBMS data. The Pass-Through Facility uses the SAS/ACCESS interface view engine to read and write data between the SAS System and the DBMS. See Chapter 6, “SQL Procedure’s Interaction with SAS/ACCESS Software,” on page 61 for detailed information.

Note: It is recommended that you use the new SAS/ACCESS LIBNAME statement to access your DBMS data more directly and to take full advantage of Version 7 enhancements. See “SAS/ACCESS LIBNAME Statement” on page 24 for more information about the new LIBNAME statement.

In the following example, you want just a brief listing of the companies to whom you have sent invoices, the amount of the invoices, and the dates on which the invoices were sent. This example accesses ORACLE data.

First, you specify a PROC SQL CONNECT statement to connect to a particular ORACLE database that resides on a remote server. You refer to the database with the alias MYDB.

Then you list the columns that you want to select from the ORACLE tables in the PROC SQL SELECT clause. In this query, you use the SAS names specified in the column-list that follows the table alias (as t1(invnum,billedon,aminus,name)). You use this column-list to rename the ORACLE columns in the query’s output.
You specify the columns in the column-list in the same order in which they appear in the SELECT statement in the Pass-Through query, shown in italics, so that the columns map one-to-one. When you specify the new SAS names in the first SELECT statement, you can specify the names in any order. Add the NOLABEL option to the query to display the renamed columns.

The PROC SQL SELECT statement uses a CONNECTION TO component in the FROM clause to retrieve data from the ORACLE table. The Pass-Through query (in italics) is enclosed in parentheses and uses ORACLE column names. This query joins data from the INVOICE and CUSTOMERS tables by using the BILLEDTO column, which references the primary key column CUSTOMERS.CUSTOMER. In this Pass-Through query, ORACLE can take advantage of its keyed columns to join the data in the most efficient way; it then returns the processed data to the SAS System.

Note: The order in which processing occurs is not the same as the order of the statements in the example. The first SELECT statement (the PROC SQL query) displays and formats the data that is processed and returned to the SAS System by the second SELECT statement (the Pass-Through query).

options linesize=120 nolabel;

proc sql;
connect to oracle as mydb (user=scott orapw=tiger
path='myorapath');
%put &sqlxmsg;

   title 'Brief Data for All Invoices';
   select invnum, name, billedon format=date9.,
        amtinus format=dollar20.2
   from connection to mydb
      (select invoicenum, billedon, amountinus,
       name
       from invoice, customers
       where invoice.billedto=customers.customer
       order by billedon) as t1(invnum,billedon,
        amtinus,name);

%put &sqlxmsg;

disconnect from mydb;
quit;

The SAS %PUT statement writes the &SQLXMSG macro variable to the SAS log so that you can check it for error codes and descriptive information from the PROC SQL Pass-Through Facility. (See Chapter 5, “Macro Variables and System Options,” on page 55 for more information.) The DISCONNECT statement terminates the ORACLE connection, and QUIT ends the SQL procedure. Output 13.1 on page 181 shows the results of the Pass-Through query.
To change the Pass-Through query into a PROC SQL view, you add a PROC SQL CREATE VIEW statement to the query. You also remove the ORDER BY clause from the CONNECTION TO component and add it to a separate SELECT statement that prints only the new PROC SQL view. Generally, it is more efficient to sort data only when needed by the program.*

```
libname slib 'Your-SAS-data-library';
proc sql;
connect to oracle as mydb (user=scott orapw=tiger path='myorapath');
%put &sqlxmsg;
create view slib.brief as
select invnum, name, billedon format=date9.,
   amtinus format=dollar20.2
from connection to mydb
   (select invoicenum, billedon,
    amountinus, name
    from invoice, customers
    where invoice.billedto=customers.customer)
   as t1(invnum,billedon,amtinus,name);
%put &sqlxmsg;
disconnect from mydb;
options ls=120 label
title 'Brief Data for All Invoices';
select * from slib.brief
order by billedon;
```

* If you have data that is usually sorted, it is more efficient to keep the ORDER BY clause in the Pass-Through query and let the DBMS sort the data.
The output from the SLIB.BRIEF view is the same as shown in Output 13.1 on page 181.
When a PROC SQL view is created from a Pass-Through query, the query’s DBMS connection information is stored with the view. Therefore, when you reference the PROC SQL view in a SAS program, you automatically connect to the correct database, and you retrieve the most current data in the DBMS tables.

Using a Pass-Through Query in a Subquery

The next example shows how to use a subquery that contains a Pass-Through query. A subquery is a nested query and is usually part of a WHERE or HAVING clause. A subquery is contained in parentheses and returns one or more values to the outer query for further processing.

Note: This example uses a Version 6 view descriptor with the Pass-Through Facility to access DBMS data. In Version 7, you can associate a libref directly with your DBMS data and use the libref in your Pass-Through query just as you would use any SAS data set. As a result, you can now create a PROC SQL view, DATA step view, or SAS/ACCESS view with DBMS data.

For this example you create a view descriptor, VLIB.ALLEMP, based on SYBASE data. The outer PROC SQL query retrieves data from the view descriptor; the subquery uses a Pass-Through query to retrieve data. This query returns the names of employees who earn less than the average salary for each department. You can use the macro variable, DEPT, to substitute the department name more easily in the query.

SYBASE objects, such as table names and columns, are case sensitive. The WHERE clauses in PROC ACCESS SUBSET statements are passed to SYBASE exactly as you type them, so you must use the correct case for SYBASE column names. The database identification statements and column names in all other statements are converted to uppercase unless they are enclosed in quotes.

```sas
proc access dbms=sybase;
/* create access descriptor */
create work.employee.access;
   server=server1;
   database=personnel;
   user=carmen;
   password=aria;
   table=employees;
/* create vlib.allemp view */
create vlib.allemp.view;
   select all;
   format empid 6.0
      salary dollar12.2
      jobcode 5.0
      hiredate date9.
      birthdate date9. ;
   list all;
```
run;

proc sql stimer;
title "Employees Who Earn Below the &dept Average Salary";

connect to sybase(server=server1
database=personnel user=carmen
password=aria);
%put &sqlxmsg;

%let dept='ACC%';

select empid, lastname, firstnam
from vlib.allempl
where dept like &dept and salary <
(select avg(salary)
from connection to sybase
(select SALARY from EMPLOYEES
where DEPT like &dept));
%put &sqlxmsg;
disconnect from sybase;
quit;

When a PROC SQL query contains subqueries or inline views, the innermost query is evaluated first. In this example, data is retrieved from the SYBASE EMPLOYEES table and returned to the subquery for further processing. Notice that the Pass-Through query is enclosed in parentheses (in italics) and another set of parentheses enclose the entire subquery.

When a comparison operator such as < or > is used in a WHERE clause, the subquery must return a single value. In this example, the AVG summary function returns the average salary of employees in the department, $57,840.86. This value is inserted in the query, as if the query were written:

```
select empid, lastname, firstnam
from vlib.allempl
where dept like &dept and salary < 57840.86;
```

Summary functions cannot appear in a WHERE clause, so using a subquery is often a good technique.

Employees who earn less than the department's average salary are returned in Output 13.2 on page 183.

**Output 13.2** Output from a Pass-Through Query in a Subquery

<table>
<thead>
<tr>
<th>EMPID</th>
<th>LASTNAME</th>
<th>FIRSTNAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>123456</td>
<td>VARGAS</td>
<td>CHRIS</td>
</tr>
<tr>
<td>135673</td>
<td>HEMESLY</td>
<td>STEPHANIE</td>
</tr>
<tr>
<td>423286</td>
<td>MIFUNE</td>
<td>YUKIO</td>
</tr>
<tr>
<td>457232</td>
<td>LOVELL</td>
<td>WILLIAM</td>
</tr>
</tbody>
</table>
In this example, it might appear to be more direct to omit the Pass-Through query and just to access VLIB.ALLEMP a second time in the subquery, as if the query were written:

```sas
%let dept='ACC%';
proc sql stimer;
  select empid, lastname, firstname
  from vlib.allemp
  where dept like &dept and salary <
    (select avg(salary)
      from vlib.allemp
      where dept like &dept);
quit;
```

However, as the SAS log in Output 13.3 on page 184 indicates, the PROC SQL query with the Pass-Through subquery performs better. (The STIMER option on the PROC SQL statement provides statistics on the SAS System’s process.)

**Output 13.3** SAS Log Comparing the Two PROC SQL Queries

```sas
213 %let dept='ACC%';
214 select empid, lastname, firstname
215 from vlib.allemp
216 where dept like &dept and salary <
217 (select avg(salary)
218   from connection to sybase
219     (select SALARY from EMPLOYEES
220      where DEPT like &dept));
221 NOTE: The SQL Statement used 0:00:00.2 real 0:00:00.20 cpu.
222 %put &sqlxmsg;
223 disconnect from sybase;
224 NOTE: The SQL Statement used 0:00:00.0 real 0:00:00.0 cpu.
225 NOTE: The PROCEDURE SQL used 0:00:00.0 real 0:00:00.0 cpu.
226 %let dept='ACC%';
227 proc sql stimer;
228 NOTE: The SQL Statement used 0:00:00.0 real 0:00:00.0 cpu.
229 select empid, lastname, firstname
230 from vlib.allemp
231 where dept like &dept and salary <
232 (select avg(salary)
233   from vlib.allemp
234   where dept like &dept);
235 NOTE: The SQL Statement used 0:00:06.0 real 0:00:00.20 cpu.
```