The PROC SQL Pass-Through Facility

DB2 Pass Through Query

FNAME	LNAME	CLAIMDT	CLAIMS
ANN	BECKER	01JAN1999:00:00:00	2003
CHRIS	DOBSON	13FEB2001:00:00:00	100
ALLEN	PARK	24SEP2001:00:00:00	10392
BETTY	JOHNSON	28NOV2001:00:00:00	3832
NANCY	PAUL	01JUN2001:00:00:00	1202

SYSTEMS SEMINAR CONSULTANTS, INC.

Presented by: Steve First

2997 Yarmouth Greenway Drive, Madison, WI 53711

Phone: (608) 278-9964 • Web: www.sys-seminar.com

The PROC SQL Pass-Through Facility

Topics:

- Overview
- Syntax
- Options
- Specifying Columns
- Row Subsetting
- Creating Views
- Creating SAS Datasets
- Date Constants and Conversion
- Error Messages

PROC SQL Pass-Through Facility

The SQL Procedure Pass-Through Facility communicates with the DBMS through the SAS/ACCESS engine.

The Pass-Through Facility allows you to:

- pass native DBMS SQL statements to a DBMS
- display the query results formatted as a report
- create SAS datafiles and views from query results

PROC SQL Pass-Through Facility

The CONNECT clause allows you to communicate with the DBMS directly.

Syntax:

PROC SQL
<CONNECT TO *dbms-name* <**AS** *alias*><
<(connect-statement-argument-1=value
...<connect-statement-argument-n=value>)>>
<(dbms-argument-1=value
...<dbms-argument-n=value>)>>;
SELECT *column-list* *<=== SAS query*
FROM CONNECTION TO *dbms-name|alias*
(select to from dbms-query) *<=== DBMS Query*
optional PROC SQL clauses;
<**DISCONNECT FROM** *dbms-name|alias;*>
<**QUIT**;>

Notes:

- The *(DBMS-QUERY)* is passed to the DBMS unchanged except for macros.

PROC SQL Pass-Through Facility

In a Pass-Through you specify:

- which DBMS to connect to
- DBMS passwords and options
- a SELECT FROM CONNECTION TO:
- a SELECT subquery that is passed directly to the DBMS enclosed in parentheses
- a DISCONNECT statement.

Notes:

- The SELECT FROM CONNECTION allows you to specify SAS formats, labels etc.
- The subquery is passed to the DBMS unchanged except for macro substitution.

PROC SQL Pass-Through Options For DB2 (partial)

SSID=DB2-subsystem-id	DB2 subsystem id.
SERVER=DRDA-server	DRDA server
AUTHID=authorization-id	DB2 authorization id

Example:

```
proc sql ;
connect to db2(ssid=ssc1);
. . .
```

Note:

- These options are the same as the ones for the LIBNAME statement, except they are coded in a different place.

A DB2 PROC SQL Pass-Through Example

Produce a report of all fields and 5 rows.

```
title 'DB2 Pass Through Query';
proc sql outobs=5;
connect to db2(ssid=ssc1);
  select *                      /* sas select           */
  from connection to db2
(select   *                     /* start pass-thru query*/
   from ssctrain.benefits       /* authid and table     */
 );                             /* end  pass-thru query */
%put &sysdbrc &sysdbmsg;        /* return codes         */
disconnect from odbc;
quit;
```

Notes:

- Everything inside the parentheses refers to DB2 names, values. (BLUE)
- Everything outside the parentheses is SAS. (RED)

Resulting Output

DB2 Pass Through Query

FNAME	LNAME	CLAIMDT	CLAIMS
ANN	BECKER	01JAN1999:00:00:00	2003
CHRIS	DOBSON	13FEB2001:00:00:00	100
ALLEN	PARK	24SEP2001:00:00:00	10392
BETTY	JOHNSON	28NOV2001:00:00:00	3832
NANCY	PAUL	01JUN2001:00:00:00	1202

Merging a DBMS Table with a SAS Dataset (DB2)

We read in a list of empcodes, most of which should match our employee file.

```
libname db2lib db2 ssid=ssc1 authid=ssctrain;
data empcode;
 input EMPNUMBER $3.;
datalines;
127
177
199
200
216
265                          <=== doesn't match
;
run;
```

Merging a DBMS Table with a SAS Dataset (DB2)

Now merge empcode with the DB2 table.

```
data subemp;
 merge db2lib.employee(in=onemp)
       empcode(in=oncode);
 by empnumber;
 if oncode and onemp;
run;
proc print data=subemp;
 title 'Selected Employee Records';
run;
```

Notes:

- The entire DBMS table is returned to SAS (inefficient).
- The DBMS will sort the EMPLOYEE table if necessary.

The Resulting Output

Selected Employee Records

Obs	FNAME	LNAME	STORENO	EMPNUMBER
1	JACK	KELLER	33312	127
2	CHRIS	FOSTER	31381	177
3	LINDA	GILBERT	33312	199
4	LYNN	GOGGIN	35618	200
5	BRENDA	WASHINGTON	31381	216

An Efficiency Problem

Merging a very small SAS file with a very large DBMS table can download entire tables only to discard most rows because they don't match.

Is there some way to tell SAS, only to bring down matching rows?

The DBKEY Option

Read a second file only if keys match.

Syntax:

data *sasdataset;*
set *sasdataset1;*
set *sasdataset2***(dbkey=***index***) key=dbkey;**

Notes:

- Read every row of *sasdataset1,* then directly read *sasdataset2* by a key value.
- *sasdataset1* must contain key values to locate in *sasdataset2*.
- _IORC_ is set to 0 if matching row is found in *sasdataset2*.

The DBKEY Option

Match the two files from the previous example.

```
options errors=0;      /* don't log not found records */
data subemp;
 set empcode;
 set db2lib.employee(dbkey=empnumber)
       key=dbkey;
 if _iorc_=0 then      /* match ? */
   output;             /* output  */
run;
proc print data=subemp;
 title 'Selected Employee Records';
 var fname lname storeno empnumber;
run;
```

Notes:

- Only matching records are retrieved from the DBMS table (efficient).
- The DBKEY variable may or may not be an index on the DBMS.

The Resulting Output

Selected Employee Records

Obs	FNAME	LNAME	STORENO	EMPNUMBER
1	JACK	KELLER	33312	127
2	CHRIS	FOSTER	31381	177
3	LINDA	GILBERT	33312	199
4	LYNN	GOGGIN	35618	200
5	BRENDA	WASHINGTON	31381	216

Note:

- DBKEY should only be used when SAS dataset is very small.

Joining a DBMS Table with a SAS Dataset (DB2)

Can we join the EMPCODE SAS dataset to EMPLOYEE table with PROC SQL?

Yes, gather the EMPCODE dataset as before.

```
libname db2lib db2 ssid=ssc1 authid=ssctrain;
data empcode;
 input EMPNUMBER $3.;
datalines;
127
177
199
200
216
265                    <=== doesn't match
;
run;
```

Joining a DBMS Table with a SAS Dataset (DB2) (continued)

Join the EMPLOYEE table with the EMPCODE SAS dataset.

```
proc sql;
  create table work.subemp as
  select a.fname,
         a.lname,
         a.storeno,
         a.empnumber
  from db2lib.employee a,
       work.empcode b
  where a.empnumber = b.empnumber
   ;
quit;
proc print data=work.subemp;
  . . .
```

Notes:

- The entire DBMS table is sometimes downloaded to SAS (expensive)

The Resulting Output

Selected Employee Records

Obs	FNAME	LNAME	STORENO	EMPNUMBER
1	JACK	KELLER	33312	127
2	CHRIS	FOSTER	31381	177
3	LINDA	GILBERT	33312	199
4	LYNN	GOGGIN	35618	200
5	BRENDA	WASHINGTON	31381	216

DBKEY Passes Keys of Rows to Select

DBKEY can also be used with PROC SQL to download only matching rows.

```
proc sql;
  create table work.subemp as
  select a.fname,
         a.lname,
         a.storeno,
         a.empnumber
  from db2lib.employee(dbkey=empnumber) a,
       work.empcode b
  where a.empnumber = b.empnumber
   ;
quit;
proc print data=work.subemp;
. . .
```

Notes:

- This is much more efficient when SAS dataset is small.

The Resulting Output

Selected Employee Records

Obs	FNAME	LNAME	STORENO	EMPNUMBER
1	JACK	KELLER	33312	127
2	CHRIS	FOSTER	31381	177
3	LINDA	GILBERT	33312	199
4	LYNN	GOGGIN	35618	200
5	BRENDA	WASHINGTON	31381	216

Debugging and Optimizing Queries

OPTIONS SASTRACE and SASTRACELOG will show generated SQL.

Examples:

```
options sastrace=',,,d';
options sastraceloc=saslog;
```

or

```
options debug=dbms_select;
```

Notes:

- debug=dbms_select shows only the select statements generated.

Debugging and Optimizing Queries (continued)

Sort a MSA table and keep only the first and last name.

```
libname msalib odbc dsn=mdbtest;

options sastrace=',,,d' sastraceloc=saslog;

proc sort data=msalib.benefits
          out=benefit2(keep=lname fname) nodupkey;
  by lname fname;
run;

proc print data=benefit2;
 title 'Benefit2';
run;
```

The Generated Log

The trace doesn't contain much of interest on this page.

```
994  proc sort data=msalib.benefits
995              out=benefit2(keep=lname fname) nodupkey;
996    by lname fname;
997  run;

TRACE: Successful connection made, connection id 4 4
   1342061696 no_name 0 SORT
TRACE: Database/data source: mdbtest 5 1342061696
   no_name 0 SORT
TRACE: AUTOCOMMIT is NO for connection 4 6 1342061696
   no_name 0 SORT
TRACE: Using FETCH for file benefits on connection 4 7
   1342061696 no_name 0 SORT
TRACE: Change AUTOCOMMIT to YES for connection id 4 8
   1342061696 no_name 0
```

The Generated Log (continued)

All of the columns were selected.

```
TRACE: SQL stmt execute on connection 4:   SELECT * FROM
   `benefits` 9
1342061696 no_name 0 SORT
TRACE: SQL stmt prepared on statement 0, connection 0
   is:  SELECT  `LNAME`, `FNAME`, `CLAIMDT`, `CLAIMS`
   FROM `benefits`  10 1342061696 no_name 0 SORT


TRACE: DESCRIBE on statement 0, connection 0. 11
   1342061696 no_name 0 SORT


NOTE: 13 observations with duplicate key values were
   deleted.
NOTE: There were 39 observations read from the data set
   MSALIB.benefits.
```

How could this be written more efficiently?

A Better Program

Keeping only the columns needed as an input option will result in a shorter query.

```
libname msalib odbc dsn=mdbtest;

options sastrace=',,,d' sastraceloc=saslog;

proc sort data=msalib.benefits(keep=lname fname)
            out=benefit2 nodupkey;
  by lname fname;
run;

proc print data=benefit2;
 title 'Benefit2';
run;
```