ADVANCED SQL AND DATABASE CONNECTIVITY
C. Michael Whitney, USAF Environmental Technical Application Center

ABSTRACT
This paper will discuss SAS® 6.07’s SQL Pass-Through Facility and how it eliminates the need for access and view descriptors when retrieving from and updating DB2® databases. Techniques described here can be applied to other relational databases. Several of SAS SQL’s other features will also be covered, including SOL and macro variables, summary functions, the case expression, and multi-table selects.

INTRODUCTION
The U.S. Air Force Environmental Technical Application Center has several hundred DB2 tables that are accessed on a daily basis. In earlier versions of SAS, we were forced to manually create a new access and view descriptor for each table we wanted to use with SAS. With the release of SAS 6.07, we were freed of this slow drudgery. The tool that liberated us is the SAS’s new Pass-Through Facility, allowing us to communicate directly to the DB2 databases without the need for the old access and view descriptors. Also, SQL has the added bonuses of being able to manipulate a lot of the data in a wide variety of ways without writing extra SAS code.

This tutorial assumes that the reader has at least a basic understanding of the SAS SQL language.

All of the techniques discussed here can be used on SAS datasets as well as DB2 tables. The word ‘table’ can be used interchangeably with ‘dataset’.

THE PASS-THROUGH FACILITY
There are five components of the Pass-Through Facility. I’ll briefly describe them below, and then give an example of how they are used.

CONNECT Statement - establishes a connection to the relational database system.
DISCONNECT Statement - cuts the link forged with the Connect statement.
EXECUTE Statement - sends SQL statements to the database.
CONNECTION TO component - allows SAS to use the database information within an SQL query.
Return Codes - two macro variables &SQLXRC and &SQLXMSG that hold the return code and error message from the database system.

Let’s create a SAS dataset called SCOTTWNO, which contains all year, month, day, hour, wind speed and direction variables from the DB2 database for Scott AFB surface observations, between 1971 and 1991.

PROC SQL NOPRINT;
CONNECT TO DB2(SSID=DSN);
CREATE TABLE SCOTTWNO AS
SELECT * FROM CONNECTION TO DB2
(SELECT YEAR, MO, DAY, HR, WSPD, WDIR
FROM ADB.SFC724338
WHERE YEAR BETWEEN 1971 AND 1991);
%PUT &SQLXRC &SQLXMSG;
QUIT;

The table will look like this:

<table>
<thead>
<tr>
<th>YEAR</th>
<th>MO</th>
<th>DAY</th>
<th>HR</th>
<th>WSPD</th>
<th>WDIR</th>
</tr>
</thead>
<tbody>
<tr>
<td>1971</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>6</td>
<td>125</td>
</tr>
<tr>
<td>1991</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>8</td>
<td>120</td>
</tr>
<tr>
<td>1971</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>4</td>
<td>125</td>
</tr>
</tbody>
</table>

As you can see, it’s standard SQL, with some extra lines added. Here’s a breakdown of what is different:

CONNECT TO DB2(SSID=DSN); tells SAS to connect to the relational database system. SSID is the SubSystem ID. If you leave it blank, it will default to whatever value is in the SAS system option DB2SSID = is set.

SELECT * FROM CONNECTION TO DB2 gets the information from the connection we’ve created. We can either use the catch-all asterisk, which returns the column names from DB2 as they exist in DB2, or we can specify new variable names by specifying them with the AS command: SELECT YEAR, MO AS MONTH, DAY, HR AS HOUR, WSPD AS WINDSPD, WDIR AS WINDDIR FROM CONNECTION TO DB2. When using the asterisk, SAS truncates any DB2 variable name that is longer than eight characters to eight. Thus, if we had asked for ELEVATION as well, it would become ELEVATIO in our SAS dataset.

(SELECT YEAR, MO, DAY, HR, WSPD, WDIR FROM ADB.SFC724338 WHERE YEAR BETWEEN 1971 AND 1991); is a standard SQL select, being passed to DB2. The FROM clause specifies the DB2 table by name.

%PUT &SQLXRC &SQLXMSG; These two macro variables hold the return codes from the relational database system. It’s wise to always include this line in your code for debugging reasons.

DISCONNECT FROM DB2; severs the connection with the database.

Now, we’ll make a new DB2 table holding precipitation data for Scott AFB, IL. It will hold year, month, day, hour, precipitation amount in tenths of inches, and precipitation type.

PROC SQL NOPRINT;
CONNECT TO DB2(SSID=DSN);
EXECUTE
(CREATETABLE ADB.SCOTT_PRECIP
(YEAR NUMERIC, MO NUMERIC,
The EXECUTE( ... ) BY DB2 command passes the SQL command to the database, telling it to create the new ADB.SCOTT_PRECIP table, and then load it with data—probably not the best way to load a lot of data, but it works.

**SUMMARY FUNCTIONS**

SQL can perform a lot of the same functions that PROC MEANS can. These functions include:

- `AVG`, `MEAN` - finds the average/mean of values
- `COUNT`, `FREQ`, `N` - finds the number of non-missing values
- `CSS` - finds the corrected sum of squares
- `CV` - finds the coefficient of variation in percent
- `MAX` - finds the largest value
- `MIN` - finds the smallest value
- `NMISS` - finds the number of missing values
- `PRT` - finds the probability of a greater absolute value of Student's t
- `RANGE` - finds the range of values
- `STD` - finds the standard deviation
- `STDERR` - finds the standard error of the mean
- `SUM` - finds the sum of the values
- `T` - finds Student's t value
- `USS` - finds the uncorrected sum of the squares
- `VAR` - finds the variance

These functions work against the entire column. If you wanted the total number of observations in a table, you'd use `SELECT COUNT(*)`. To get the maximum value, you'd use `MAX(column-name)`. DB2 returns the value name of EXPRESSN to SAS when using the summary functions. If multiple summaries are used, you'll get `EXPRESSN1, EXPRESSN2, EXPRESSN`, etc. Use the `AS` command to rename them to something more useful.

We're interested in finding out what the average wind speed and direction is for Scott AFB. So we use the following code, and specify what to call the averages returned:

```
PROC SQL NOPRINT;
CONNECT TO DB2(SSID=DSN);
CREATE TABLE SCOTTAVG AS
SELECT EXPRESSN AS AVGWSPD,
     EXPRESSO AS AVGWDIR
FROM ADB.SFC724338;
%PUT &SQLXRC &SQLXMSG;
DISCONNECT FROM DB2;
QUIT;
```

The output would look like this having one observation with two variables:

<table>
<thead>
<tr>
<th></th>
<th>AVGWSPD</th>
<th>AVGWDIR</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>6</td>
<td>128</td>
</tr>
</tbody>
</table>

But, what if we wanted to know the monthly average by year for these two?

```
PROC SQL NOPRINT;
CONNECT TO DB2(SSID=DSN);
CREATE TABLE SCOTTAVG AS
SELECT YEAR, MO, EXPRESSN AS AVGWSPD,
     EXPRESSO AS AVGWDIR
FROM ADB.SFC724338
GROUP BY YEAR, MO;
%PUT &SQLXRC &SQLXMSG;
DISCONNECT FROM DB2;
QUIT;
```

Here the GROUP BY clause is used to specify that we want the averages grouped by each value in our YEAR and MO columns. The resulting dataset would look like this:

<table>
<thead>
<tr>
<th>YEAR</th>
<th>MO</th>
<th>AVGWSPD</th>
<th>AVGWDIR</th>
</tr>
</thead>
<tbody>
<tr>
<td>1971</td>
<td>1</td>
<td>6</td>
<td>120</td>
</tr>
<tr>
<td>1971</td>
<td>2</td>
<td>10</td>
<td>117</td>
</tr>
<tr>
<td>1971</td>
<td>3</td>
<td>12</td>
<td>119</td>
</tr>
</tbody>
</table>

**MACRO VARIABLES**

Macro variables can be used anywhere within the SQL code. SQL can also place values into macro variables for you.

The following code example shows how you can make your SQL code more flexible by using macro variables within it:

```
%LET DATATBl = %STR(SCOTTWND);
%LET SLCTSTR = %STR(YEAR, MO, DAY, HR, WDIR, WSPD);
%LET TBLNAME = %STR(ADB.SFC724338);
%LET WHERESTR = %STR(WHERE YEAR BETWEEN 1971 AND 1991);
PROC SQL NOPRINT;
CONNECT TO DB2(SSID=DSN);
CREATE TABLE &DATATBl AS
    SELECT &SLCTSTR
    FROM &TBLNAME
    WHERE &WHERESTR;
%PUT &SQLXRC &SQLXMSG;
DISCONNECT FROM DB2;
QUIT;
```
We can use this same piece of code with different tables and variables by simply changing the macro variable assignments. Note that &WHERESTR could be left blank, thus giving us all of the data in the DB2 table.

Now, let's say we want to find the minimum and maximum winds for Scott AFB, and the days they occurred. Two SQL statements will be used to do this for us. The first uses the MIN and MAX summary functions, and places them into &MINWND and &MAXWND. The second will return the year, month, day and hour of whenever the minimum and maximum winds occur.

```sas
PROC SOL NOPRINT;
CONNECT TO DB2(SSID=DSN);
SELECT MIN, MAX INTO :MINWND, :MAXWND
FROM CONNECTION TO DB2
(SELECT MIN(WSPD), MAX(WSPD)
FROM ADB.SFC724338);
%PUT &SQLXRC &SQLXMSG;
DISCONNECT FROM DB2;
QUIT;
```

```sas
PROC SOL NOPRINT;
CONNECT TO DB2(SSID=DSN);
CREATE TABLE MINMAX AS
SELECT YEAR, MO, DAY, HR, WSPD, WDIR
FROM ADB.SFC724338
WHERE WSPD = &MINWND OR
WSPD = &MAXWND;
%PUT &SQLXRC &SQLXMSG;
DISCONNECT FROM DB2;
QUIT;
```

Data from this table is not shown, however, the output from the first would be &MINWND holding 0, and &MAXWND holding 49. Output from the second would be all observations where the winds were either 0 or 49.

Note that in the first SQL statement, you must specify what values you are getting from DB2, instead of using the catch-all asterisk. In SAS/SQL, the INTO command will place the values returned into macro variables, whose names must be preceded with a colon, i.e., :MINWND and :MAXWND. Here we list all of the variables, followed by the INTO, and then the names of the macro variables we want the values put into.

In our second SQL code, we create a dataset called MINMAX that contains the dates, wind speeds and directions of any observations that match our min and max.

**SELECTING FROM MULTIPLE TABLES**

Often you will need to get information from two or more different tables. This can easily be accomplished with SQL. In the following code, we will get the forecast wind direction from ADB.TAF724338, and what actually was observed for that hour from the ADB.SFC724338 table.

```sas
PROC SQL NOPRINT;
CONNECT TO DB2(SSID=DSN);
CREATE TABLE SCOTTZ AS
SELECT YEAR, MO, DAY, HR, TAFWDIR, WDIR AS OBSWDIR
FROM CONNECTION TO DB2
(SELECT A.YEAR, A.MO, A.DAY, A.HR, A.TAFWDIR, B.WDIR
FROM ADB.TAF724338 AS A, ADB.SFC724338 AS B
WHERE A.YEAR = B.YEAR AND A.MO = B.MO AND A.DAY = B.DAY AND A.HR = B.HR);
%PUT &SQLXRC &SQLXMSG;
DISCONNECT FROM DB2;
QUIT;
```

Output would look like this:

<table>
<thead>
<tr>
<th>YEAR</th>
<th>MO</th>
<th>DAY</th>
<th>HR</th>
<th>TAFWDIR</th>
<th>OBSWDIR</th>
</tr>
</thead>
<tbody>
<tr>
<td>1971</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>120</td>
<td>120</td>
</tr>
<tr>
<td>1971</td>
<td>1</td>
<td>3</td>
<td>120</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

We're renaming the variables in the first SELECT statement, because WDIR is also part of the second table.

Our first SELECT statement will plug those returned wind directions from the second table into the new variable called OBSWDIR.

In the code being passed to DB2, we use table aliases to specify the columns we are getting from each of the two DB2 tables. The line from ADB.TAF724338 AS A, ADB.SFC724338 AS B sets up the aliases using the AS operator. Our passed SELECT statement tells SQL that we want year, month, day, hour, and wdir from the ADB.TAF724338 table, and wdir from the ADB.SFC724338 table, where the dates match up in the two. Please note, that you'll only get data back when the dates match...you won't get any data from either table when the dates DON'T match up.

A multi-table select can be across many tables, but, the more tables you query, the longer the SQL takes to run.

**THE CASE EXPRESSION**

The case expression is used to select result values when certain conditions are met. It can be used to group values. The format of the case expression is:

```sas
CASE <case-operand>
WHEN when-condition THEN
result-expression
<WHEN when-condition THEN
result-expression>
<ELSE result-expression>
END
```

The WHEN-THEN clauses are used to evaluate the condition, and give the result action for that condition. The ELSE option is the catch-all for anything that doesn't meet your tests. When the case-operand is specified, it is compared to the when-condition for equality. If they are equal, then the WHEN clause is true and the result-expression occurs. If the case-operand isn't specified after the CASE, it must be specified in the WHEN filed.
CASE TEST
  WHEN 1 THEN 'YES'
  WHEN 0 THEN 'NO'
  ELSE 'MAYBE'
END
or
CASE
  WHEN TEST=1 THEN 'YES'
  WHEN TEST=2 THEN 'NO'
  ELSE 'MAYBE'
END

Let's say we want to group surface winds at Scott AFB into several categories: Calm, Mild, Strong, and Severe. The case expression lets us do this without an extra SAS data step:

PROC SQL NOPRINT;
CONNECT TO DB2(SSID=DSN);
CREATE TABLE WINDGRP AS
  SELECT YEAR, MO, DAY, HR, WDIR, WSPD,
  CASE
    WHEN WSPD < 2 THEN 'CALM'
    WHEN WSPD < 15 THEN 'MILD'
    WHEN WSPD < 55 THEN 'STRONG'
    ELSE 'SEVERE'
  END AS CATEGORY
  FROM CONNECTION TO DB2
    (SELECT YEAR, MO, DAY, HR, WDIR, WSPD,
     FROM ADB.SFC724338);
%PUT &SQLXRC &SQLXMSG;
DISCONNECT FROM DB2;
QUIT;

Note that we had to put the CASE statement in the local SELECT, and not in the DB2 SELECT. Also, this shows an example of how an expression can be renamed just like a column/variable name, using the AS command. This piece of code will give us a dataset with year, mo, day, hour, wdir, wspd and category.

Output would look like this:

<table>
<thead>
<tr>
<th>YEAR</th>
<th>MO</th>
<th>DAY</th>
<th>HR</th>
<th>WSPD</th>
<th>WDIR</th>
<th>CAT</th>
</tr>
</thead>
<tbody>
<tr>
<td>1971</td>
<td>1</td>
<td>1</td>
<td>6</td>
<td>125</td>
<td>120</td>
<td>CALM</td>
</tr>
<tr>
<td>1971</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>50</td>
<td>150</td>
<td>STRONG</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

CONCLUSIONS

The SAS SQL Pass Through Facility is a very powerful tool, that allows easy access to your relational databases. It's much faster to use than creating specific access and view descriptor to do the same work, and has the added benefit of not having to be stored separately like the descriptors do.

REFERENCES
