Some Less Known but Powerful Proc SQL Features
Jun Zhao, Boehringer Ingelheim Pharmaceuticals, Inc. Ridgefield, CT

Abstract

This paper will introduce four less known but useful Proc SQL features. They are the MIX join, the outer join with where clause, the multi-level CASE expression, and the statistics through Boolean expression. This paper will also give sample code to demonstrate the usage of these features.

Introduction

Proc SQL in SAS® is a powerful tool for data querying. However, some useful SQL features are not widely known, which limits its use. This paper will discuss various features discovered through personal experience. Codes and examples are used to demonstrate how to use these features and achieve processing efficiency.

The MIX Joins

It is widely known that SQL allows inner joins of up to 16 tables. In addition, SQL also permits outer joins of multiple tables through a series of two table outer joins. Furthermore, there is a lesser known SQL feature that gives the user power to mix inner and outer joins in one SQL step. This is referred to as a MIX Join.

For instance, you have tables A, B and C with variables A1 & A2, B1 & B2, and C1 & C2 respectively, and you want to inner join tables A and B with the result being left join with C. The convention way would be using one SQL step to perform the inner join, followed by a second SQL step to perform the left join with C; or one could use an in-line view to inner join A and B and a left join the result with C. However, this can be accomplished through one step using the MIX join as follows:

```
PROC SQL;
SELECT A.*, B.*, C.*
FROM A INNER JOIN B
ON A1=B1
LEFT JOIN C
ON C1=A1
QUIT;
```

Here, we simply treat a inner join as one kind of outer join, like left join, right join or full join.

Outer Join with WHERE Clause

An outer join is used in conjunction with the ON clause where as an inner join is used with the WHERE clause. However, if your left join necessitates subsetting of the left dataset, the ON clause would not suffice. For example, you have the following two tables A and B:

Data A:

<table>
<thead>
<tr>
<th></th>
<th>X1</th>
<th>X2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>

Data B:

<table>
<thead>
<tr>
<th></th>
<th>Y1</th>
<th>Y2</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>

First you want to join A and B upon \(X_1=Y_1\) and \(X_2=Y_2\) and include every row in table A. You can simply use the following left join:

```sql
PROC SQL;
SELECT A.*, B.*
FROM A LEFT JOIN B
ON X1=Y1 AND X2=Y2;
QUIT;
```

It will result the following output:

<table>
<thead>
<tr>
<th>X1</th>
<th>X2</th>
<th>Y1</th>
<th>Y2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>5</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>3</td>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>4</td>
<td>.</td>
<td>.</td>
</tr>
</tbody>
</table>

Next, if you decide to exclude \(X_1=1\) in table A, you could try to use this code:

```sql
PROC SQL;
SELECT A.*, B.*
FROM A LEFT JOIN B
ON X1=Y1 AND X2=Y2 AND X1>1;
QUIT;
```

As you can see, the result doesn't change. The reason is that by definition when SAS performs a left join, every record in the left table will be included. One can always subset table A in a previous step before the left join with B. But one possible solution is to combine the ON and WHERE clause as follows:

```sql
PROC SQL;
SELECT A.*, B.*
FROM A LEFT JOIN B
ON X1=Y1 AND X2=Y2
WHERE X1>1;
QUIT;
```

As you can see from the result, the \(X_1=1\) group is excluded from the output. Interestingly enough, if you want to subset data from table B, it would be possible to accomplish this by utilizing the ON clause as follows:

```sql
PROC SQL;
SELECT A.*, B.*
FROM A LEFT JOIN B
ON X1=Y1 AND X2=Y2 AND Y1^=4;
QUIT;
```

### Multi-level Case Expression

Case expression is used for conditional execution in SQL. It's usage is straightforward when used with a one level condition such as IF \(A=B\) THEN \(X=Y\). However, multi-level conditions require a more complex usage of the case expression.

For instance, you have a multi-level condition as follows:
IF I<=10 THEN DO;
    IF K<=20 THEN DO;
        IF J<30 THEN E=1;
    ELSE E=2;
    END;
ELSE IF 30<=K<=40 THEN DO;
    IF J<30 THEN E=7;
    ELSE E=8;
    END;
ELSE DO;
    IF J<30 THEN E=9;
    ELSE E=10
    END;
ELSE IF 1<=20 THEN DO;
    IF J<30 THEN E=3;
    ELSE E=4;
    END;
ELSE THEN DO;
    IF J<=30 THEN E=5;
    ELSE E=6;
    END;
END;

To use single level CASE expression, it must be coded as follows:

PROC SQL;
    CREATE TABLE C AS
    SELECT *
    CASE
        WHEN 1<=10 AND J<30 AND K<=20 THEN 1
        WHEN 1<=10 AND J>=30 AND K<=20 THEN 2
        WHEN 1<=10 AND J<30 AND 40>=K>=30 THEN 7
        WHEN 1<=10 AND J>=30 AND 40>=K>=30 THEN 8
        WHEN 1<=10 AND J<30 AND (20<K<30 OR K >40) THEN 9
        WHEN 1<=10 AND J>=30 AND (20<K<30 OR K >40) THEN 10
    END AS E
    FROM A;
QUIT;

However, there is an equivalent yet more efficient method: the multi-level CASE expression:

PROC SQL;
    CREATE TABLE B AS
    SELECT *
    CASE
        WHEN 1<=10 THEN CASE
            WHEN K<=20 THEN CASE
                WHEN J<30 THEN 1
            ELSE 2
            END
        WHEN 40>=K>=30 THEN CASE
            WHEN J<30 THEN 7
        ELSE 8
            END
        ELSE CASE
            WHEN J<30 THEN 9
        ELSE 10
            END
        END
    WHEN 1<=20 THEN CASE
        WHEN J<30 THEN 3
    ELSE 4
        END
    ELSE CASE
        WHEN J<30 THEN 5
    ELSE 6
        END
    END AS E
    FROM A;
QUIT;

There are many advantages in using the multi-level CASE. As noted in the lines marked above, the ELSE statement can be used to assign values when conditions are not met. This is not always possible in the single level CASE expression where you are required to list all the possible conditions. When there are many complex conditions, a multi-level CASE expression is logically simpler to code, and more efficient to execute than a single level expression. The efficiency increases with the size of the dataset and the complexity of the condition.

Statistics Through Boolean Expression

It is not an easy task to conditionally count or summarize grouped data using SQL. In the above sample Table A, if you want to know the number of X2 values greater than X1 in each X1 group alone with the percentage, you could utilize the following code which performs the
summarization based upon a given Boolean expression

PROC SQL;
SELECT X1, SUM(X2>X1) AS N LABEL='N OF X2 > X1',
    MEAN(X2>X1) AS PCT LABEL='Pt X2 > X1'
FROM A
GROUP BY X1;

We have the expected result:

<table>
<thead>
<tr>
<th>X1</th>
<th>N OF X2 &gt; X1</th>
<th>Pct OF X2 &gt; X1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3</td>
<td>100.00%</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>66.70%</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>33.30%</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>0.00%</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>0.00%</td>
</tr>
</tbody>
</table>

Using the same technique, we can also calculate the total value of those X2’s which are greater than X1 in each X1 group alone with their mean values.

PROC SQL;
SELECT X1,
    SUM((X2>X1)*X2) AS T LABEL='TOTAL VALUE OF X2 > X1',
    (CALCULATED T)/SUM(X2>X1) AS M LABEL='MEAN OF X2>X1'
FROM A
GROUP BY X1;

Conclusion

SQL users will appreciate the power that SQL provides, especially when it comes to querying and joining data. However, many of its features are not well known. Through the creative use of SQL, users can discover many surprising techniques. Therefore enhancing the power of SQL and broadening its use.

Trademarks

SAS is registered trademark of SAS Institute, Inc. in the USA and other countries.

Acknowledgment

Thanks to Boehringer Ingelheim Pharmaceuticals, Inc. for supporting me in the writing and presenting this paper. Special Thanks to my colleagues Susan Italiano and Sheldon Wong for their help with editing.

Reference


Author

The author welcomes comments and suggestions:

Jun Zhao
Biometrics and Data Management
Boehringer Ingelheim Pharmaceuticals, Inc.
900 Ridgebury Rd./P.O.Box 368
Ridgefield, CT 06877-0368
☎: (203)798-4819