Paper 226-2011

LAG Function Combined with Conditional Functions – Useful in Identifying Differences in Like Data

Andy Hummel, Delta Air Lines, Atlanta, GA

ABSTRACT

The LAG function is useful in identifying subtle differences in rows with similar data. This is especially valuable when the data set contains a large number of rows. Additionally, when conditional functions are used in conjunction with the LAG function specific limits can be used to flag only particular differences between rows.

INTRODUCTION

The LAG function can return a value from a previous row of data, or compare the current row value to a previous row. LAG can be used to look back 1 or more than 1,000 rows depending on the programmer's needs. When LAG is used in combination with conditional functions such as IF, AND, OR, =, or NOT = it becomes a powerful evaluator of duplicate data. This paper will demonstrate applied uses of LAG in combination with conditional functions to flag duplicate rows of data.

Data that is manually entered into a database can often contain duplicate and inconsistent data. This is especially true when the data is entered by multiple users in a dynamic environment. Duplicate and conflicting records can lead to redundant expenses; such as: a hotel room booked in two different cities for the same night and employee.

The data used in the following examples was manually entered into the database by multiple coordinators who set up hotel stays for employees. Due to a number of factors including multiple hotel requests from the same employee, irregular operational issues and user error the database may contain discrepancies. The following SAS® examples will show how to flag the discrepancies. The variables used in the data set are: "employee number", "city name", "hotel name" and "night of hotel stay".

STEP 1: SORT THE DATA

The first and most critical step is to correctly sort the data based on the variables that the LAG function will evaluate. In this example we want to evaluate Empl_Nbr, Airport_City and Hotel_Name. We will sort Empl_Nbr first since this is the primary key, followed by Airport_City and Hotel_Name since they are the secondary keys.

```
/* SORTING THE DATA */
PROC SORT DATA=raw_hotel_data OUT=hotel_check_1;
      BY empl_nbr airport_city hotel_name;
RUN;
```

Table 1. Data Sorted by Empl_Nbr, Airport_City, Hotel_Name

Empl_Nbr	Airport_City	Hotel_Name	date_5_4_2010
1741	ATL	Bobs Best Hotel	X
1741	ATL	Crabby Inn	X
2292	BWI	Crabby Inn	X
2786	BWI	Crabby Inn	X
3413	BWI	Crabby Inn	X
3792	ATL	Bobs Best Hotel	X
3876	ATL	Crabby Inn	X
3876	BWI	Crabby Inn	X
4379	ATL	Bobs Best Hotel	X
4379	ATL	Crabby Inn	X
5083	BWI	Crabby Inn	X
5298	ATL	Bobs Best Hotel	X
5712	BWI	Crabby Inn	X
6359	ATL	Bobs Best Hotel	X
6807	BWI	Crabby Inn	X
6920	ATL	Crabby Inn	X
6920	BWI	Crabby Inn	X
7335	ATL	Bobs Best Hotel	X
7335	BWI	Crabby Inn	X
8241	ATL	Bobs Best Hotel	X
9218	BWI	Crabby Inn	X
9240	BWI	Crabby Inn	X
9827	ATL	Bobs Best Hotel	X
9959	ATL	Bobs Best Hotel	X

STEP 2: EVALUATE FOR SAME EMPLOYEE, SAME CITY, DIFFERENT HOTEL

Here the LAG function is used to evaluate the data to determine if an employee is booked in the same city but in different hotels. In this example the Empl_nbr and Airport_Name in the current row must match the previous row, conversely Hotel_Name in the current row must be different than Hotel_Name in the previous row in order for the row to be flagged. The matching rows are highlighted in the below table.

```
DATA hotel_check_2;
       SET hotel_check_1;

       IF empl_nbr = LAG(empl_nbr)
              AND airport_city = LAG(airport_city)
              AND hotel_name  NE LAG(hotel_name)
       THEN same_city_diff_hotel='Yes';
       ELSE same_city_diff_hotel='No';
RUN;
```

Table 2. Same Employee, Same City, Different Hotel

Empl_Nbr	Airport_City	Hotel_Name	date_5_4_2010	same_city_diff_hotel
1741	ATL	Bobs Best Hotel	X	No
1741	ATL	Crabby Inn	X	Yes
2292	BWI	Crabby Inn	X	No
2786	BWI	Crabby Inn	X	No
3413	BWI	Crabby Inn	X	No
3792	ATL	Bobs Best Hotel	X	No
3876	ATL	Crabby Inn	X	No
3876	BWI	Crabby Inn	X	No
4379	ATL	Bobs Best Hotel	X	No
4379	ATL	Crabby Inn	X	Yes
5083	BWI	Crabby Inn	X	No
5298	ATL	Bobs Best Hotel	X	No
5712	BWI	Crabby Inn	X	No
6359	ATL	Bobs Best Hotel	X	No
6807	BWI	Crabby Inn	X	No
6920	ATL	Crabby Inn	X	No
6920	BWI	Crabby Inn	X	No
7335	ATL	Bobs Best Hotel	X	No
7335	BWI	Crabby Inn	X	No
8241	ATL	Bobs Best Hotel	X	No
9218	BWI	Crabby Inn	X	No
9240	BWI	Crabby Inn	X	No
9827	ATL	Bobs Best Hotel	X	No
9959	ATL	Bobs Best Hotel	X	No

STEP 3: REVERSE SORT TO CAPTURE THE MATCHING ROW

In the previous example, where the same employee was booked in two different hotels, the second of each matching row was flagged with a “YES”.

Empl_Nbr	Airport_City	Hotel_Name	date_5_4_2010	same_city_diff_hotel
1741	ATL	Bobs Best Hotel	X	No
1741	ATL	Crabby Inn	X	Yes

By visually looking at the data a user could easily identity the flagged rows if the data set is small, but visual inspection is not efficient or practical for large data sets.

If we take the original data set and sort hotel_name in DESCENDING order and then run the lag comparison again we can capture the other half of our matching rows. When using the DESCENDING statement option, place “DESCENDING” prior to the variable you wish to sort in DESCENDING order.

```
/* SORTING THE DATA WITH "hotel_name" IN DESCENDING ORDER*/
PROC SORT DATA=lag_data.lag_source OUT=hotel_check_3;
       BY empl_nbr airport_city DESCENDING hotel_name;
RUN;
```

After the data is sorted, run the lag evaluation step again.

```
DATA hotel_check_4;
       SET hotel_check_3;

       IF empl_nbr = LAG(empl_nbr)
              AND airport_city = LAG(airport_city)
              AND hotel_name  NE LAG(hotel_name)
       THEN same_city_diff_hotel='Yes';
       ELSE same_city_diff_hotel='No';

RUN;
```

Table 3.

Empl_Nbr	Airport_City	Hotel_Name	date_5_4_2010	same_city_diff_hotel
1741	ATL	Crabby Inn	X	No
1741	ATL	Bobs Best Hotel	X	Yes
2292	BWI	Crabby Inn	X	No
2786	BWI	Crabby Inn	X	No
3413	BWI	Crabby Inn	X	No
3792	ATL	Bobs Best Hotel	X	No
3876	ATL	Crabby Inn	X	No
3876	BWI	Crabby Inn	X	No
4379	ATL	Crabby Inn	X	No
4379	ATL	Bobs Best Hotel	X	Yes

We now have the second half of our same employee, same city but different hotel rows. Notice that in Table 2 the row with “Crabby Inn” was “Yes” while in Table 3 (the descending row data) the row with “Bobs Best Hotel“ is now “Yes”. We now have both parts of our flagged rows and can create one data set with both rows.

STEP 4: COMBINE MATCHING ROWS

Now we can create one data set that combines both halves of our matching rows. Additionally, we can also discard any rows that did have a match.

The below code creates a new data set called “hotel_check_5” from “hotel_check_2”, which had “hotel_name” sorted in ASCENDING order, and “hotel_check_4”, which had “hotel_name” sorted in DESCENDING order.

Since the matching rows had “same_city_diff_hotel” flag set to “Yes” we can include an IF statement to OUTPUT only the “Yes” rows.

The PROC SORT statement will sort the rows by “empl_nbr” so that is easier to visually evaluate the data.

```
DATA hotel_check_5;
      SET hotel_check_2 hotel_check_4;

      IF same_city_diff_hotel='Yes' THEN OUTPUT;

      PROC SORT;
      BY empl_nbr;
RUN;
```

Table 4. All Matching rows

Empl_Nbr	Airport_City	Hotel_Name	date_5_4_2010	same_city_diff_hotel
1741	ATL	Crabby Inn	X	Yes
1741	ATL	Bobs Best Hotel	X	Yes
4379	ATL	Crabby Inn	X	Yes
4379	ATL	Bobs Best Hotel	X	Yes

The above table has all rows where the same employee was booked in the same city but in two different hotels for the same night. The end-user can use this data to cancel the incorrect booking which will hopefully to reduce hotel costs.

ADDITIONAL USES: LOOKING BACK MORE THAN 1 ROW

The above examples used the LAG function to evaluate the immediate preceding row of data. LAG also has the ability to look back more than just the previous row. By placing a numeric value after the LAG statement you can tell LAG how far back to look. In the below example LAG is comparing the current row to the 4th previous row since the number 4 is placed after LAG.

```
/* EVALUATING THE CURRENT ROW TO THE 4TH PREVIOUS ROW OF DATA */
IF empl_nbr = LAG4(empl_nbr)
THEN same_empl_nbr='Yes';
```

By adding the conditional function of "OR", LAG can compare the current row to multiple rows of previous data. The following example will flag the current row if the employee number in the current row matches the employee number in any of 1st through 5th previous rows. This was accomplished by placing the numbers 1, 2, 3, 4 and 5 after LAG.

```
/* EVALUATING BACK 5 ROWS OF DATA */
IF (empl_nbr = LAG1(empl_nbr))
       OR (empl_nbr = LAG2(empl_nbr))
       OR (empl_nbr = LAG3(empl_nbr))
       OR (empl_nbr = LAG4(empl_nbr))
       OR (empl_nbr = LAG5(empl_nbr))
THEN same_empl_nbr='Yes';
```

ADDITIONAL USES: USING A LOOP TO EVALUATE MULTIPLE ROWS

Often the user will not know how many rows they need to look back. In this case a %DO %TO loop can be used inside a MACRO to look back as many rows as needed.

For this example we will use an illustration from the airline industry. There are certain airports that require special training in order for a pilot to be qualified to land. If a pilot does not land at the airport every 60 days he must undergo additional training to be qualified for that airport. Our goal is to flag a pilot who is scheduled to land at a special airport but has not done so within 60 days of the scheduled landing.

Our data has the variables, "date", "airport", and "landing number at that airport". "Landing number at that airport" keeps a record of each landing at an airport. For example, the first time a pilot lands at the Las Vegas airport the "landing number at that airport" would be "1" the second landing would be "2" and so on.

The first step is to sort the data in chronological order by "date".

Table 5. Data Sorted in Chronological Order

Date	Airport	Landing_Nbr_At_Airport
3/16/2010	JAC	15
4/1/2010	BWI	8
4/2/2010	LAS	14
4/3/2010	ATL	78
4/13/2010	JAC	16
4/14/2010	ATL	79
4/19/2010	LAS	15
5/1/2010	LAX	2
5/26/2010	ATL	80
6/20/2010	JAC	17
6/21/2010	ATL	81
6/23/2010	LAS	16
6/24/2010	LAX	3
7/2/2010	JAC	18
7/7/2010	BWI	9

The special airport we are interested in is Jackson Hole, Wyoming (JAC). If the pilot is scheduled to land at JAC and has not done so in the past 60 days we want to flag that record so that the pilot receives additional training prior to an attempted landing. Since we do not know how many other landings occurred between the two JAC landings we are unsure of how many rows back LAG needs to look. If we use a %DO %TO loop inside a MACRO we can quickly evaluate as many rows as desired.

Below is the complete code which is followed by a step-by-step explanation.

```
%MACRO landing_check();

DATA airport_landing_1;
       SET airport_landing_soruce;

       %DO look_back=1 %TO 15;
              IF airport='JAC'
              AND airport=lag&look_back.(airport)
              AND landing_nbr_at_airport-1=lag&look_back.(landing_nbr_at_airport)
              AND date-lag&look_back.(date)>=60
              THEN Out_of_Range='YES';
       %END;

       RUN;

%MEND;

%landing_check();
```

First, since we know the data set has 15 rows we will set our %DO %TO to 1 and 15. By assigning the variable "look_back" a range and placing "look_back" after LAG we can define how many rows to look back without having to place a fixed number after LAG. The value in using a %DO %TO loop is that if the data set had 1,000 rows we could easily set "look_back" to 1,000 and quickly loop through al 1,000 rows. Additionally, the program can quickly be changed to meet the programmer's needs.

```
       %DO look_back=1 %TO 15;
```

We then want to set our conditionals. First we only want to evaluate the data if the airport is JAC. We do this with the below line of code.

```
       IF airport='JAC'
```

Then we want to make sure that the airport in the current row is equal to the airport in the LAG row we are evaluating.

```
       AND airport=lag&look_back.(airport)
```

Next we want to use "landing number at airport" to make sure the landing in the current row is the landing that directly followed the landing in the LAG row. We accomplish this by subtracting 1 from the current row "landing number at airport" value and making sure it equals the value in "landing number at airport" in the LAG row. Only the landing that directly preceded the landing in the current row will make this conditional true.

For Example: the landing on 6/20/2010 was landing number 17 at JAC, and the landing on 7/2/2010 was landing number 18 at JAC.

Date	Airport	Landing_Nbr_At_Airport
6/20/2010	JAC	17
7/2/2010	JAC	18

For the row with a date of 7/2/2010 the "landing number at airport" is 18, and the only previous row where

```
       landing_nbr_at_airport-1=lag&a.(landing_nbr_at_airport)
```

can be true is the landing on 6/20/2010 which was landing number 17.

Finally we evaluate the dates. If the date in the current row minus the date in the lag row is greater than 60 days the conditional is true and we flag the current row and the pilot for additional training.

```
AND date-lag&a.(date)>=60
```

In order for our conditional flag to be true all five conditional settings must be met. The table below shows that there was more than a 60 day gap between the April 13 and June 20 landings. This would notify the airline that the pilot required additional training prior to their June 20 flight.

Table 6. Evaluated Data

Date	Airport	Landing_Nbr_At_Airport	Out_of_Range
3/16/2010	JAC	15	
4/1/2010	BWI	8	
4/2/2010	LAS	14	
4/3/2010	ATL	78	
4/13/2010	JAC	16	
4/14/2010	ATL	79	
4/19/2010	LAS	15	
5/1/2010	LAX	2	
5/26/2010	ATL	80	
6/20/2010	JAC	17	YES
6/21/2010	ATL	81	
6/23/2010	LAS	16	
6/24/2010	LAX	3	
7/2/2010	JAC	18	
7/7/2010	BWI	9	

CONCLUSION

The above examples illustrated how to use the LAG function in conjunction with conditional functions as an evaluator of like data. The use of LAG with conditional functions makes this a particularly powerful tool for setting detailed and specific flags.

CONTACT INFORMATION

Your comments and questions are encouraged. Contact the author for the program and data presented in this paper at:

Andy Hummel
Delta Air Lines, Inc.
Department 028
P.O. Box 20706
Atlanta, GA 30320-6001
Work Phone: 404-715-1270
Email: Andrew.Hummel@delta.com