Paper 3298-2015

The Great Dilemma of Row-Level Permissions for LASR Tables

Emre G. SARICICEK, The University of North Carolina at Chapel Hill.

ABSTRACT

Many industries are challenged with requirements to protect information and limit its access. In this paper, we will discuss various approaches for row-level access to LASR tables and demonstrate our implementation. Methods discussed in this paper include security joins in data queries, using star schema with security table as one dimension, permission conditions based on metadata stored user information, and user IDs being associated with data as a dedicated column. The paper then identifies shortcomings and strengths of various approaches as well as our iterations to satisfy business needs that led us to our row-level permissions implementation. In addition, the paper offers recommendations and other considerations to keep in mind while working on row-level permissions with LASR tables.

INTRODUCTION

SAS® Visual Analytics is an in-memory solution for exploring large volumes of data and building reports.

The purpose of this paper is to provide examples of row-level security implementations within SAS Visual Analytics.

When you share tables with other users in an organization by giving them access to a table, by default they can access to all the data in that table. Row-level security enables us to control access to rows on a table based on the data in the row. (e.g department, unit or context)

In this paper we will focus on row-level security for a financial transactions table. We will show security join in data queries, using star schema and creating a custom column to control permission based on metadata stored user attributes.

We assume that as data builder you already know how to do data queries and have some understanding of metadata user management. Example data structures we use in this paper are a financial transactions table and an user permissions table.

Our Financial transactions tables has typical columns (keys and decriptions) for a transactions table such as Account, Activity,Budget, DepartmentID, EmployeeID, ProjectID and FiscalYear. Security tables is where we have kept which user has access to what department, project or employee. It has columns like UserID, DepartmentID, ProjectID, EmployeeID.

MANAGING SECURITY FOR LASR TABLES

We can setup condition for Lasr table security from SAS Visual Analytics Administrator. As Figure 1 shows, manage environment allows us to browse Lasr tables and we can access the authorization page for a table on navigation from right click menu of a Lasr table.

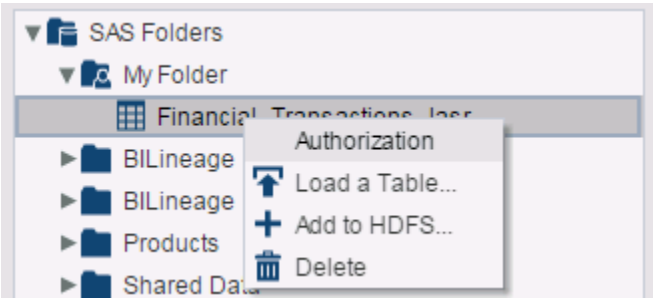

Figure 1. Authorization Selection for LASR Table

There are explicit grants or deny setup options for a Lasr table but for row-level security we are going to select conditional grant option.

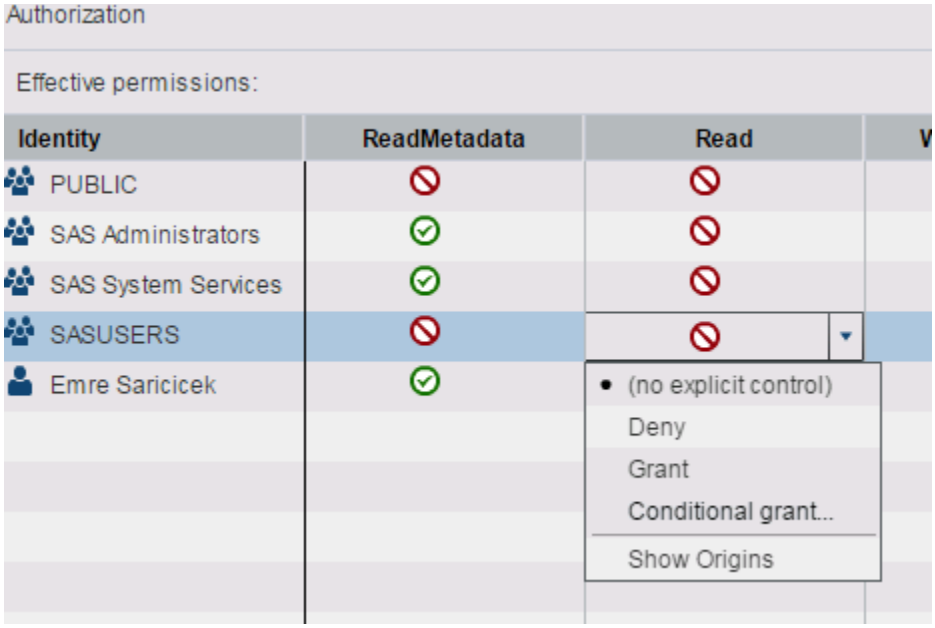

Figure 2. Authorization Page for LASR Table

We can setup conditional grant for specific groups or users that are in metadata. While setting conditions you can use identity driven properties such as PersonName, ExternalIdentity, IdentityGroupName.

Also there are other operators for permission condition. You can set that condition using other identity values such as IdentityGroups containing department_id.

With some good planning on identity driven properties you can have more flexibility on permission conditions. Some identity driven properties are not always populated in metadata.

One of those are ExternalIdentity value, which is a synchronization key in bulk user load. Also keep in mind that for this property only the first value is returned. If value is not set an empty string is returned and for properties such as Userid uppercase format is returned.

While setting permission conditions remember comparisons are case sensitive.

As Figure 3 shows, by setting permission condition here we are setting the row level permission.

In our example, group we set condition on, can only see data where employee_id column value matches ExternalIdentity value. In this case when user looking to transactions on this LASR table will only see transactions where his employee_id is present on the employee_id column.

By setting this condition we made user to see only transactions related to same user. We can also specify conditions for column values as well. If we wanted to limit this group to only be able to see certain transactions amounts we could setup filter based on that as well. Such as some groups can only see certain rows where projectid is set.

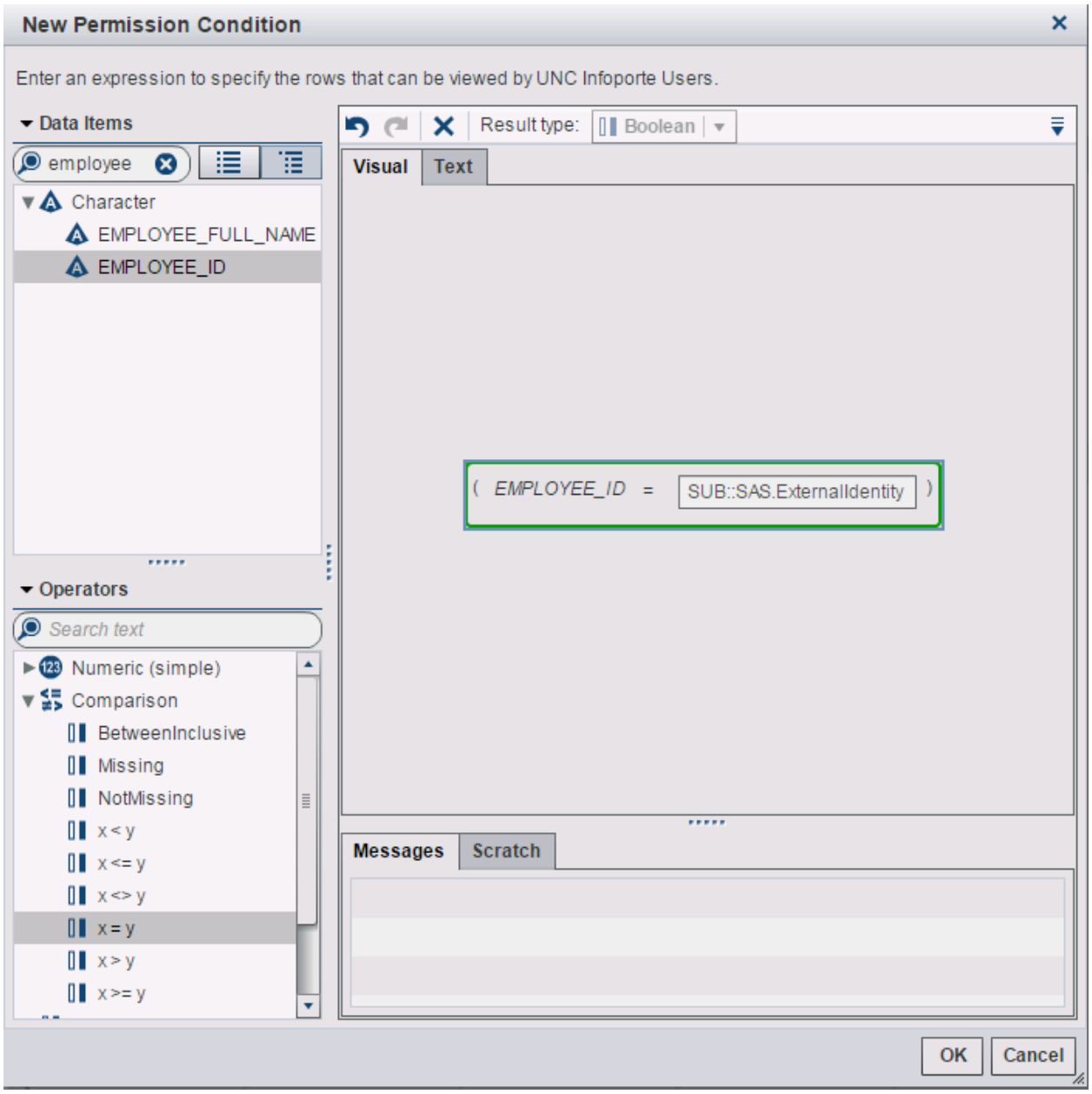

Figure 3. Permission Condition Setup

JOIN DATA IN DATA QUERIES

One way to prepare data for row-level security on LASR is doing a security join in data query. This data query can be used to pull data in to memory. Basically we are preparing a LASR table where we can have possible combinations of permissions with our main transactions table.

As you can see from Figure 4, we can end up with a Cartesian product. This may not be an issue with small sized permissions table or small transactions table but we have seen preparing data this way for some reporting tables are unmanageable. Join conditions here can be added to prevent Cartesian product but in this case it's actually a valid case such as star schema join between dimension tables.

If you know that there is only single dimension key to join on and sizes are fitting in to your available memory in LASR then creating data queries this way should not preferred over LASR star schema.

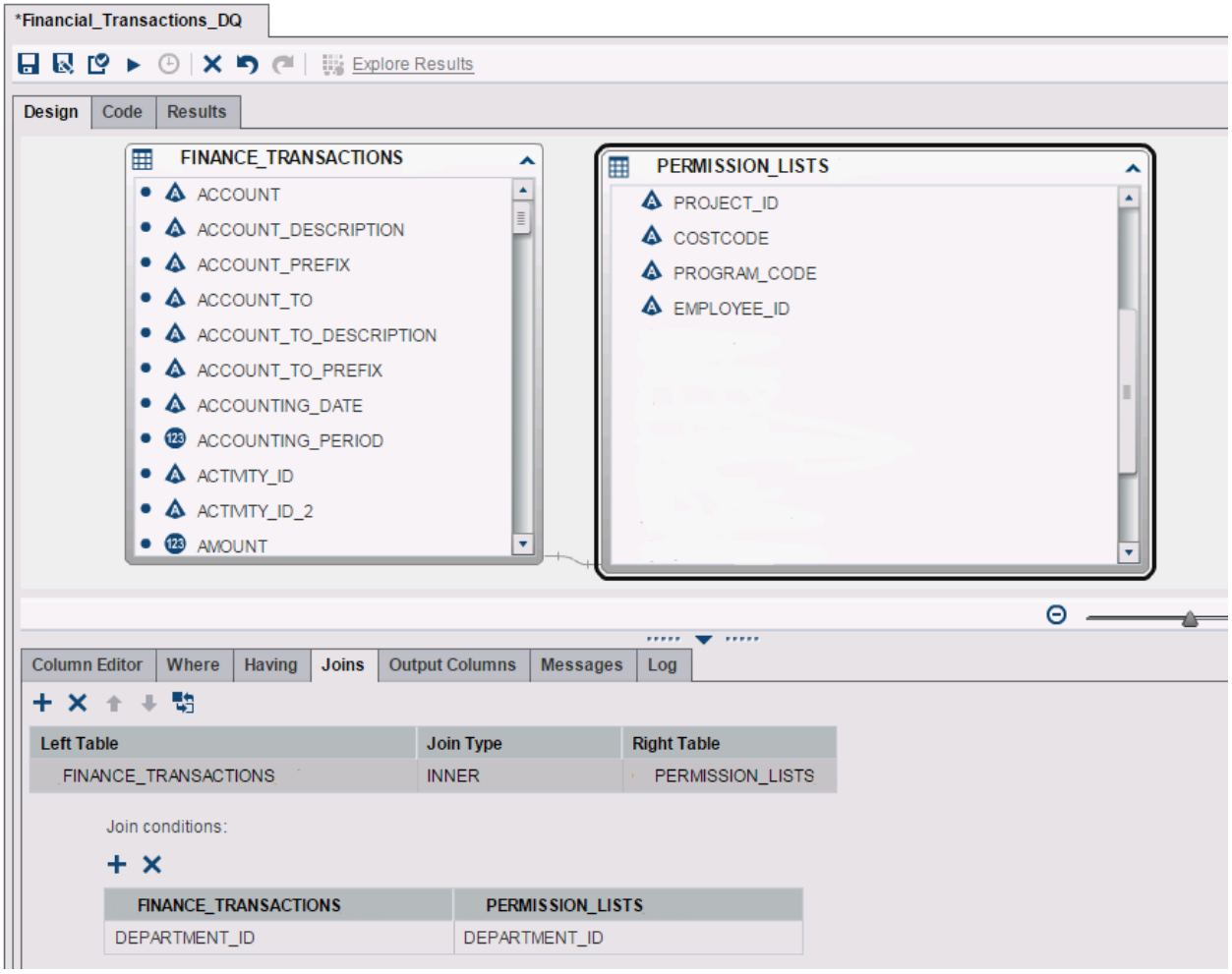

Figure 4. Data Query Join

After preparing data and loading to LASR we need to visit permission control setup for table and add needed conditions. Resulting LASR table will have all columns to set conditional grant and from Visual Analytics Administrator you can setup those including associating filter like conditions for group access.

When using SAS.Userid identity property in conditional grant screen keep in mind that SAS returns that value uppercase. For web authdomain return will be USERID@DOMAIN format.

Here is an example setup for this permission condition presented in Figure 5 as text and Figure 6 as visual.

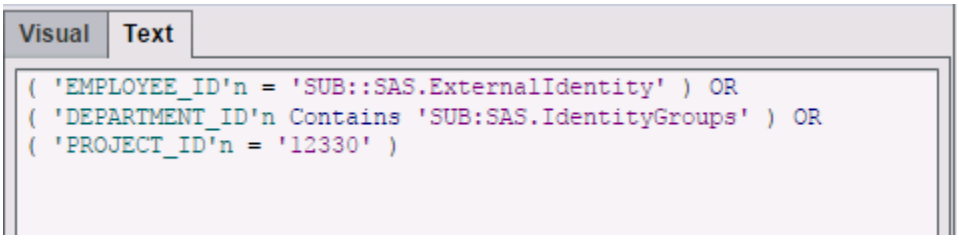

Figure 5. Conditional Permission Setup Text View

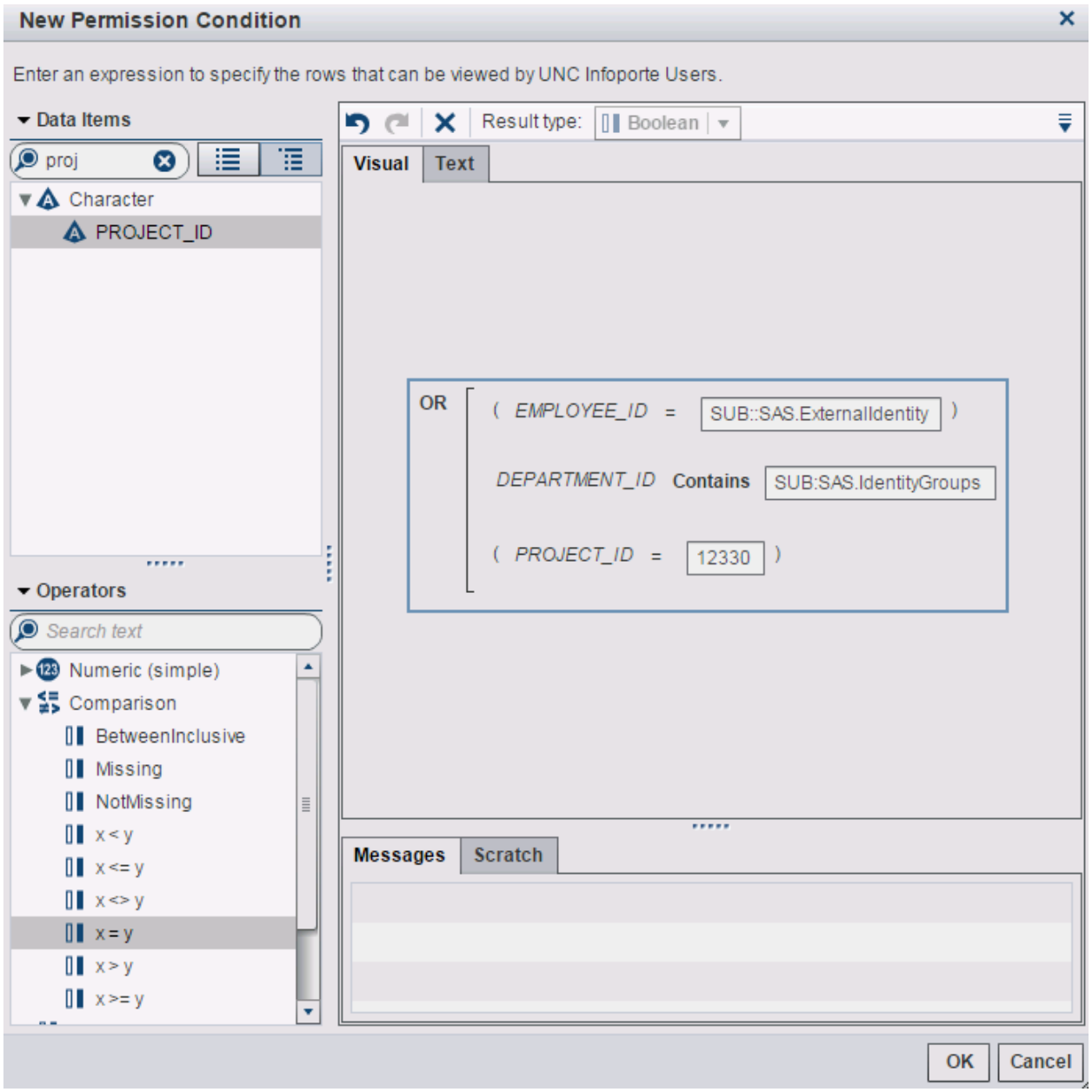

Figure 6. Conditional Permission Setup

In our case having a big permission table which had almost all campus users and their relations by many columns made this unusable for us because result LASR table was ending up being so big that we were not able to load it due to cross join. Supporting row-level security this way simply meant keeping every possible combinations of user access in relation with data. We did consider partitioning data and loading those partitioned tables as LASR tables and set conditional grants based on group level for each partitioned data. Partitioning approach quickly got off the table since there were transactions across departments and projects. Also this approach would quickly become unmanageable for data loading and conditional permissions setup process.

This made us think about joining tables in memory rather than preparing in data query and loading the result set to LASR. We started looking into creating a start schema in Visual Data Builder.

USING STAR SCHEMA IN MEMORY JOINS

SAS Visual Data Builder has an option to create a LASR star schema. Building a LASR star schema has some advantages. It requires some preparation such as input tables must be loaded to SAS LASR Analytic Server. If tables don't have same data type and length for dimension key and column, you can correct those in preparation data queries for tables you are creating LASR star schema. On output tab you can set it to be a view where SAS joins tables in memory so building LASR star simplifies security joins. This basically means you need to have individual data queries to load tables first. In our case we did create separate LASR tables for Transactions and Permissions tables.

First table you select to add to star schema generation becomes fact table. Any other table added after that becomes dimension. As you can see from Figure 7 security table becomes a dimension. Since star schema fact and dimension can be associated in one column this can be used to satisfy row-level security.

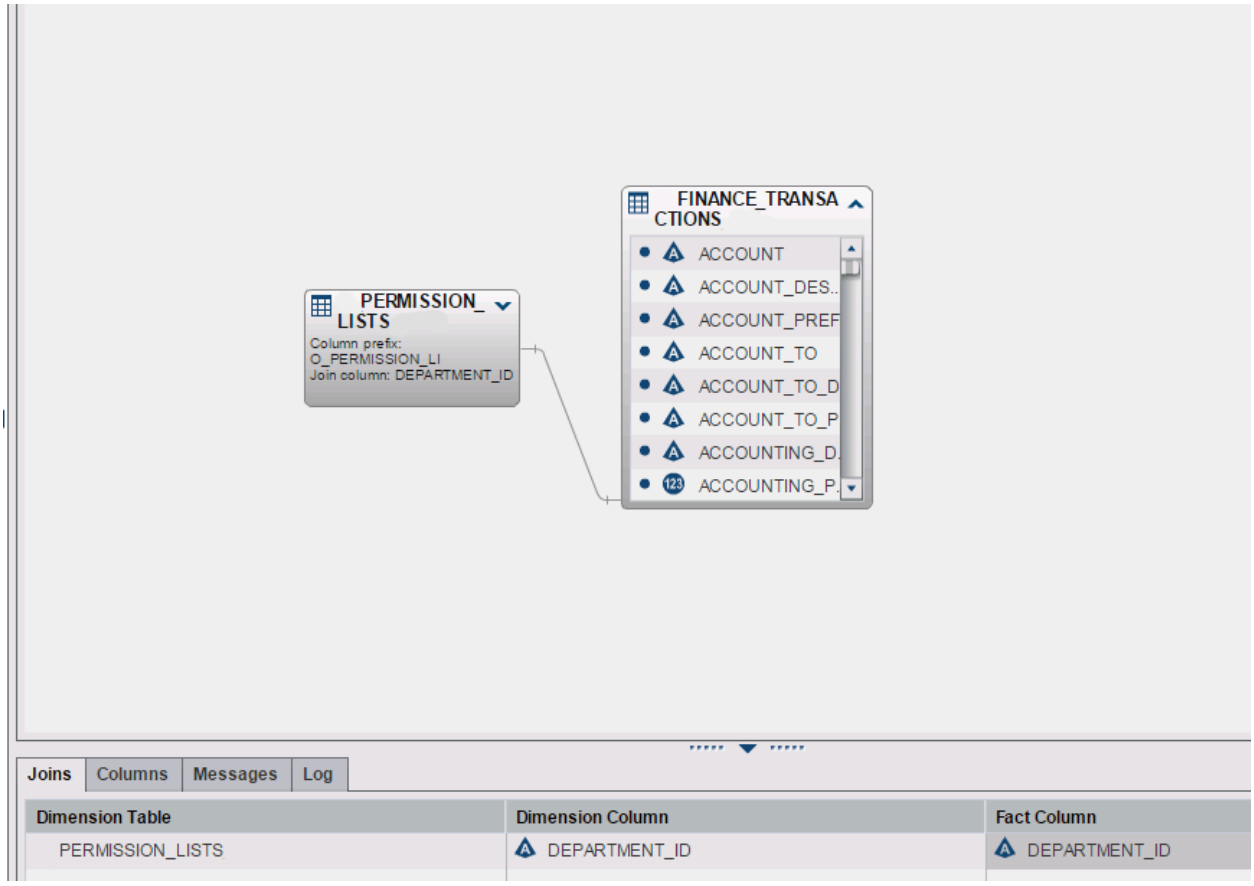

Figure 7. Star Schema LASR

To use memory more efficiently Visual Data Builder also allows star schema to be built as a view as well. LASR star schema as a view is a good feature but accessing data through this way can impact performance. This highly depends on size of the permission table in our case here. Again creating LASR star schema as table requires more physical memory. Also report filters on a LASR star schema view does not perform well. If you have memory resources for best performance create a table.

In our case LASR star schema was partially satisfying our requirements. Because it's not just on one column we were looking for security. Business required having multiple conditions for row-level access to be satisfied. Such as department_id, employee_id and project_id we were not able benefit from a star schema.

This limitation of single dimension key join forced us to think more on how to meet requirements where multiple columns needed to be taken into consideration for row-level security. We started exploring addition of userid as a column on LASR tables.

ADDITION OF USERID AS A COLUMN

We had a quite complex row-level security access requirement. We needed to determine row-level security based on data on multiple columns. Row–level security had to be evaluated based on five different columns and their combinations. Such as granting access to that row if user has access to one or one of the other focused columns on that row. We decided to create a database view joining transactions and permissions table where we determined users who can access data and created a new column and concatenated those userids separated by commas.

We added userids to column using oracle xmlagg function in database and created a view which we used in our data query to load to LASR.

UserIDs
Userid1,userid2,userid3,userid4,userid5,….

After adding a column to table where all possible users who could access to row was listed, we updated conditional grant to check for SAS.PersonName being contained in this column.

('USER_IDS'n Contains ',SUB::SAS.PersonName,')

We have used SAS identity property PersonName to hold userid values when we bulk synchronized users.

There have been many drawbacks using this approach. After creating userid column and trying to load to LASR we quickly realized text length was not enough for us to hold all userids there. Default length is 1024 and we altered this at library properties advanced options as seen on Figure 8.

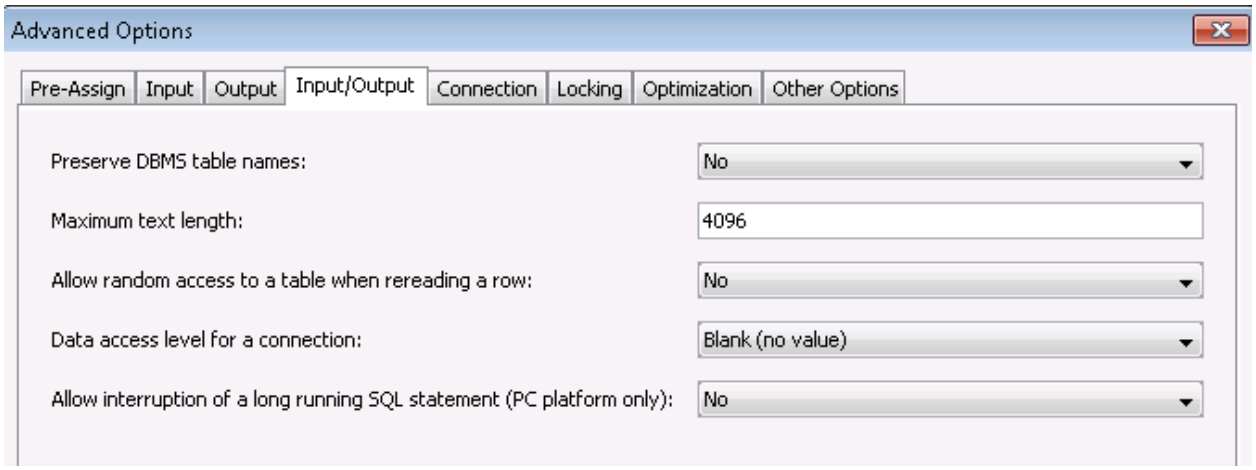

Figure 8. Advanced Library options

Our major problem has been load times of this data and how much memory is consumed. We always keep in mind that for every row in transactions we have a column with a length of 4096.

For a transactions table of 60Million rows this operation of calculating userids and placing them on column is a costly database operation. This also affects load time of that data and table size in memory easily goes beyond couple hundred GB.

Also during report viewing Visual Analytics has to evaluate contains statement for row-level security and we have seen CPU usage spikes taking longer than usual due to this condition being evaluated for every row.

Report filters in SAS Visual analytics are data driven. This means when a user loads a report to view for every filter or selection options that are set for the report is evaluated. At that point conditional grant is evaluated as well. As you can guess for every filter and options SAS has to evaluate contains statement on every transaction row. These operations are really expensive but SAS handles those really well.

CONCLUSION

We have demonstrated that we can implement row-level security in different ways. Conditional grants on Visual Analytics Administrator offers flexible filtering options on table. On tables already having a column to apply row-level security, using identity properties in conjunction with conditional grants is way to handle row-level security.

Creating LASR star schema table can significantly improve performance of reports and if single dimension key is sufficient for security join, it's a clean way to address this problem as well.

For our approach we have seen that Visual Analytics is quite powerful to handle computations on columns while accessing millions of rows at the same time. But this is quite expensive on having a lot dedicated memory. Our approach to have userids as a column is resource consuming both on memory and network load times. Also creating a computed column is another step to maintain and creation of that data itself can be resource intensive.

RECOMMENDED READING

- *SAS Institute Inc. 2014. SAS ® Visual Analytics 6.4 User's Guide Cary, NC: SAS Institute Inc.*
- *SAS Institute Inc. 2011. SAS ® 9.3 Guide to BI Row Level Permissions Cary, NC: SAS Institute Inc.*

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

EMRE G. SARICICEK
The University of North Carolina at Chapel Hill
emre@unc.edu