Paper 12640-2016

Security in Public Reports: Combining Strict Row-Level Authorization with Publically Available Reports Using SAS® Visual Analytics

Levi Lindblom, Accenture; Jonas Lie-Nielsen, SAS Institute Inc.

ABSTRACT

How can you give thousands of customers access to the same reports with the same underlying SAS LASR tables and be confident in that they only see the data that belongs to themselves? Row-level security is the functionality in SAS® Visual Analytics that solves this and is easy and straight forward in the course examples. The challenge is how to implement row-level security in a complex real-life example with a high number of users and a huge security hierarchy controlling the access rights?

This paper explains and evaluates the most common implementation alternatives based on the experience and performance testing from a customer project. It also explains how the selected alternative was implemented.

INTRODUCTION

This paper shares the project experience from an implementation of SAS Visual Analytics for a huge number of B2B customers that require quite complex row-level security setup within SAS Visual Analytics driven by strict security requirements, combined with a high number of users and concurrent sessions as well as a complex user hierarchy.

The paper will provide the reader with a general understanding of how row-level security works in SAS Visual Analytics as well as explain and evaluate the five implementation alternatives that were most relevant for the project. Finally, the paper explain how the selected alternative was implemented.

The paper is directed to solution architects and developers who needs to gain a deeper understanding in how row-level security works and want to explore the different possibilities for how to implement it. The paper expects the reader to have a basic understanding of what row-level security is and how it works.

BACKGROUND

This paper is based on the project experience from an implementation of SAS Visual Analytics reporting for B2B customers. Visual Analytics was set up as the reporting part of the B2B users “personal website” used for administrating and purchasing services.

The goal was to create a modern, robust and automated reporting solution serving all the customers. The solution should create a shared set of reports for the customers and at the same time ensuring that no users could see data that they shouldn’t. The companies using the solution varied from small companies to large multinational enterprises with more than 200 report users, where different users had access to different parts of the company.

REQUIREMENTS

The projects key delivery was a set of Visual Analytics reports available to all B2B users. Each user must only see data they have access to. The main requirements regarding row-level security were:

- New or removed users can be added at any time and needs to get access to reports within a few hours.
- Individual access rights changes continuously and needs to be reflected in Visual Analytics within a few hours.
- Some users should not have access to all reports.
- Report queries should be quick.

- Adding/removing of users and changes to a user's access rights within SAS Visual Analytics must be automated.
- Internal users should be able to see all data

CHALLENGES

The project requirement of ensuring that all users must only see the data that they had access to turned out to be quite challenging. This was due to a combination of several factors that in combination created a setup that we could not find a clear "best practice" for how to implement. The key challenges were:

- The administration of users and their access rights there distributed to one or more administrators within each B2B customer. They were able to control access rights for other users in their organization from within their "personal webpage". This could be done at any time and was stored into two custom-made solutions, one for each product line. This setup made it nearly impossible to make changes in the security rules.
- A complex security hierarchy, with the possibility of connecting users to several nodes at different levels within the security hierarchy at the same time, not only the leaf nodes (accounts). A user can be connected to one or more accounts, customer and companies (master nodes) as illustrated in Figure 1 below.

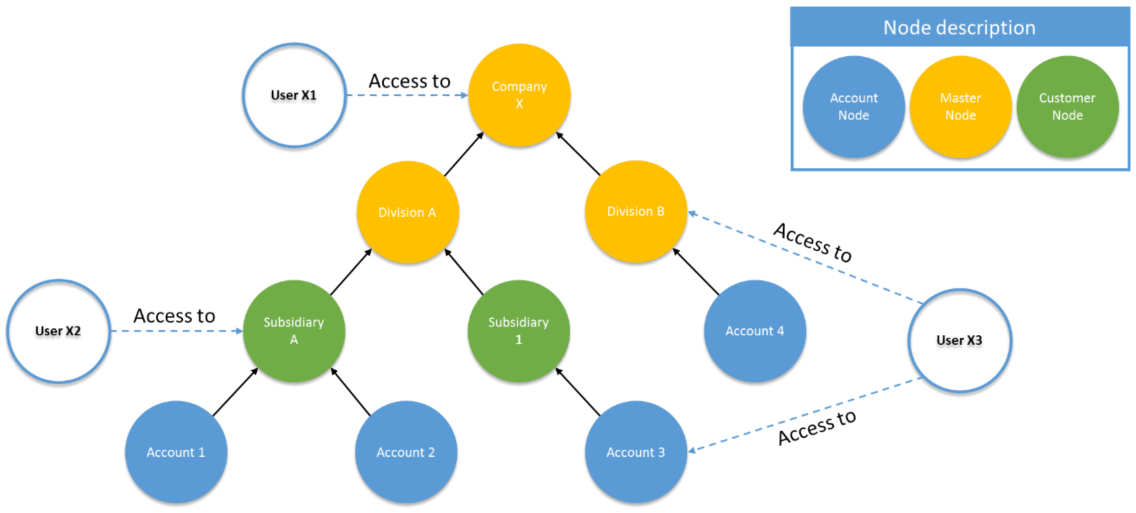
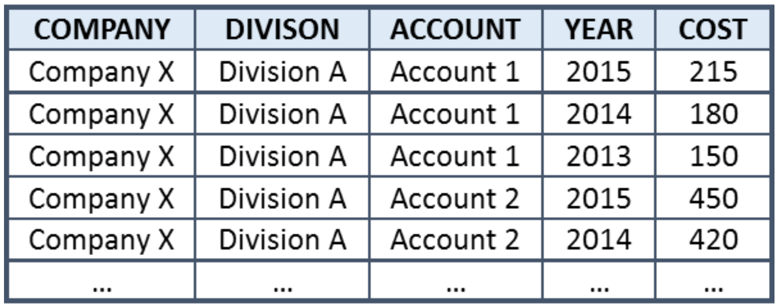


Figure 1. Security hierarchy

- A user can be given access to an account directly or indirectly by granting access to a master node or customer node and in that way get access to all underlying accounts.
- Most of the actual data were connected to a specific account (the leaf node), but some of the data were connected directly to the Master node or the Customer Node.
- A high number of users and companies, starting off at about 70 000 users within more than 35 000 companies, both expected to grow over the next few years.
- One user could have access to multiple accounts, companies and customers. There was no maximum number of resources a user could get access to but at the time of writing some users had access to more than 22 000 individual accounts.
- One account could be accessed by multiple users. The maximum was over 200 at the time of writing.

- There is more than 50 000 nodes within the company hierarchy across all the companies using the solution
- Users are continuous added through the self-service part of the “personal website” with a belonging requirement of giving them access to the reports as soon as possible.
- In addition to the users from each B2B customer, it was some internal users that should have access to all the data and all the reports
- Most data sets has multiple rows for each account as is illustrated in Figure 2:



COMPANY	DIVISON	ACCOUNT	YEAR	COST
Company X	Division A	Account 1	2015	215
Company X	Division A	Account 1	2014	180
Company X	Division A	Account 1	2013	150
Company X	Division A	Account 2	2015	450
Company X	Division A	Account 2	2014	420
...	...	...	...	...

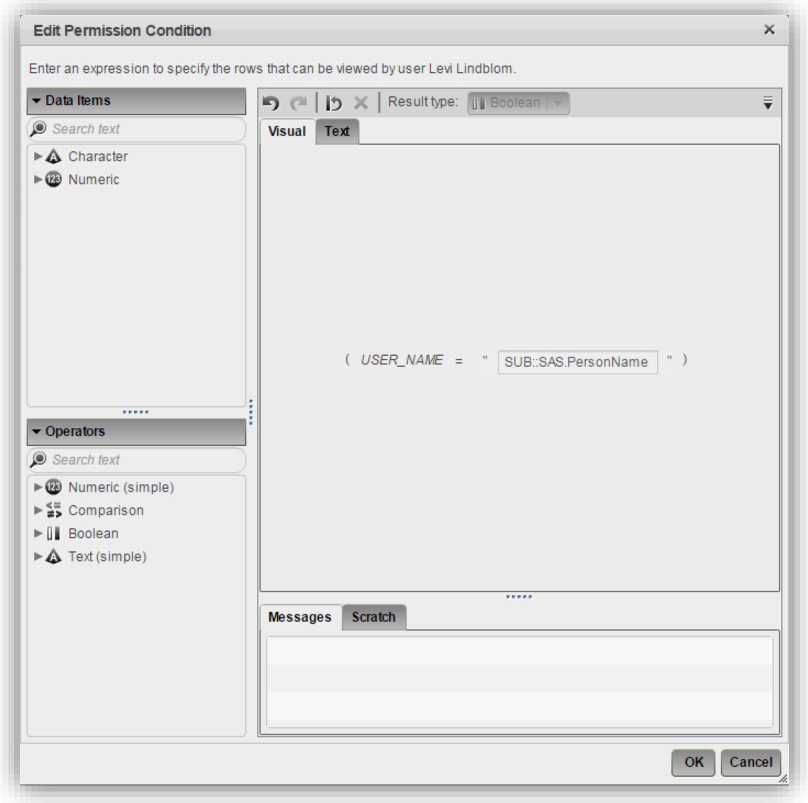
Figure 2. Sample data set

Based on the volumes and the dynamics of the solution, it was quite clear that a fully automated solution was needed, but how? Using the traditional solution from the training courses of matching the metadata group of the user to a column in the SAS LASR™ table holding the department would certainly not be good enough. Based on the characteristics and flexibility of the security hierarchy above, the only way to have an implementable and fixed rule that was possible to implement into SAS Visual Analytics in an automated way was to map all user access rights down to accounts. That means that a user is tagged into all accounts that is connected to the nodes that the user has access to.

ALTERNATIVE SOLUTIONS

SAS Visual Analytics provides functionality for row-level security that enables you to filter data in a report based on who is viewing it. Row-level security allows you to control who can read individual rows in LASR tables by setting the conditional grant permission for the group/role/user you want to only see a subset of the data.

Conditional grant permissions allow you to create Boolean expressions called permission conditions that are evaluated for each row in the LASR table, only rows evaluated to true are visible to the user. Permission conditions can use columns from the LASR table and properties on the user’s profile as part of the expression as illustrated in Figure 3 below.



Enter an expression to specify the rows that can be viewed by user Levi Lindblom.

(USER_NAME = " SUB::SAS.PersonName ")

Figure 3. Edit Permission Condition screen

Table 1 shows some of the user properties that you can use when creating permission conditions:

Property	Description
SUB::SAS.Userid	The requesting users authenticating ID
SUB::SAS.IdentityGroups	The names of groups and roles the requesting user is a part of
SUB::SAS.PersonName	The Name of the requesting user
SUB::SAS.ExternalIdentity	External identity property of the requesting user

Table 1. User properties

Row-level security functionality in Visual Analytics satisfies the main requirement of the project, to show different data to different users based on their access rights. With this functionality we can filter the data based on any of the *resources* in the node hierarchy; Account Nodes, Customer Nodes or Master nodes. In the following section we will review the five alternative solutions considered in this project.

ALTERNATIVE 1 – EXTERNAL IDENTITY

One way to implement row-level security is to save the resource IDs that a user has access to on the users' profile, for example in the External Identity property. By concatenating multiple IDs in the property, a user can get access to multiple resources. For example if a user should only see data for the HR and marketing accounts, they external identity can be set to "HR,MARKETING" and if the ACCOUNT column contains the accounts each row is associated with, a permission condition similar to "*SUB::SAS.ExternalIdentity contains ACCOUNT*" could be used on the LASR table.

This alternative is useful if each user has access to a small amount of resources so that the concatenated list of resource IDs is smaller than the maximum size of the external identity property. In this project some users has access to over 2000 different resources which meant that the concatenated list of their account IDs could not be saved in the external identity.

ALTERNATIVE 2 – METADATA GROUPS

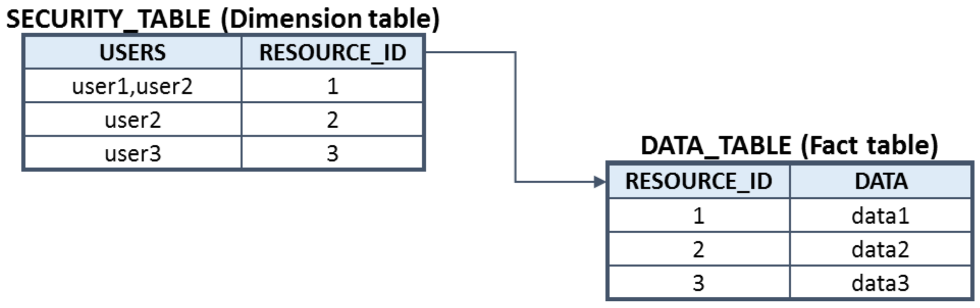
In this alternative you create a metadata group for each resource that a user can get access to. Adding or removing users from a group controls who has access to that resource. The row-level security is implementing by comparing the resource ID in the LASR table with the IdentityGroups property on the users profile which returns a concatenated list of the groups a user has access to.

This alternative can be beneficial in situations with a limited number of resources. In our project, access needs to be tracked on an account level in the hierarchy, meaning each account needed its own metadata group. This would require over 50 000 groups and the users within each to be tracked and maintained in the metadata.

ALTERNATIVE 3 – STAR SCHEMA

Both the previous alternatives are based on saving access rights on the users' metadata profile and matching it with the data in LASR. In this and the rest of the alternatives, access rights (mapping between the users and the data they can see) is saved in LASR together with the data.

In this alternative, you create a security table in LASR that contains access rights for all users. Reports are based on a LASR star schema where the data is saved in the fact table and the security table is set as a dimension table. The security table contains one row for each resource, with a column containing the concatenated list of all names/IDs of users that have access to that resource. Figure 4 illustrates this alternative on sample data.



SECURITY_TABLE (Dimension table)

USERS	RESOURCE_ID
user1,user2	1
user2	2
user3	3

DATA_TABLE (Fact table)

RESOURCE_ID	DATA
1	data1
2	data2
3	data3

Figure 4. Star schema illustration

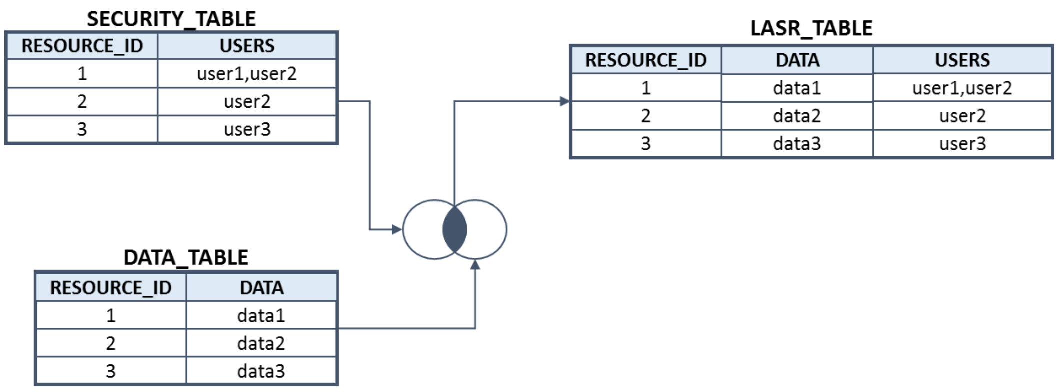
Row-level security is placed on the output table on the star schema. The conditions checks if the users name (SUB::SAS.PersonName) is contained in the concatenated user list for this row.

Reports are built on the output table of the star schema, the output table is created when a user tries to access the data. In addition, both tables are fully joined before permission conditions are checked. This leads to reduced performance compared to pre-joined tables.

Note: In this project the concatenated user list was split up in to three columns with the most used users in the first column for better performance.

ALTERNATIVE 4 – DENORMALIZED TABLE

This solution uses the same security table and permission conditions as the star schema solution but replaces the star schema with denormalized table to improve performance when reports are loaded. The data is pre-joined with the security table and the resulting denormalized table is uploaded to LASR. Since joining is not performed at runtime the loading of reports becomes quicker, however more space is needed in LASR. Figure 5 illustrates the join used to create the denormalized table.



SECURITY_TABLE

RESOURCE_ID	USERS
1	user1,user2
2	user2
3	user3

DATA_TABLE

RESOURCE_ID	DATA
1	data1
2	data2
3	data3

LASR_TABLE

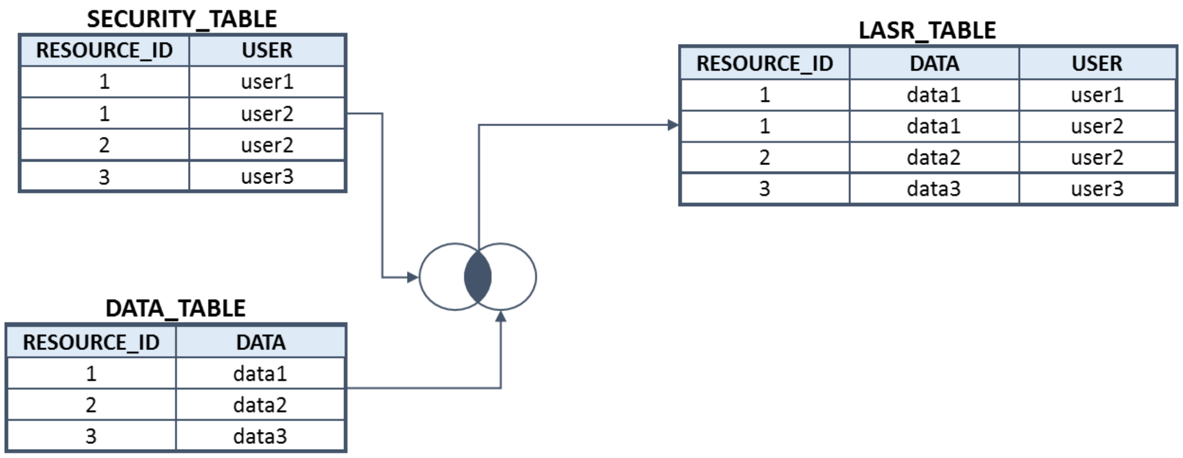
RESOURCE_ID	DATA	USERS
1	data1	user1,user2
2	data2	user2
3	data3	user3

Figure 5. Denormalized table illustration

A major issue with this alternative is that it might fail if enough users gets access to the same resource. If the total length of the USERS column is exceeded by the addition of a new user, row-level security would not work properly. In this case either the length of the USERS column needs to be extended, or an additional column added. This issue is also applicable for the star schema alternative.

ALTERNATIVE 5 – DATA DUPLICATION

The last solution modifies the security table used in the previous solutions. Instead of keeping all users with access to a resource on one line in the security table, it keeps one mapping of resource – user per line. Consequently, the security table will have multiple rows for each resource ID from the data table. When joining the security table and a data table, using the resource ID as the join key, all rows will be duplicated for each user that have access to them. The permission conditions can be set to check if the USER_ID column matches the requesting user name. Figure 6 shows this functionality on sample data.



SECURITY_TABLE

RESOURCE_ID	USER
1	user1
1	user2
2	user2
3	user3

DATA_TABLE

RESOURCE_ID	DATA
1	data1
2	data2
3	data3

LASR_TABLE

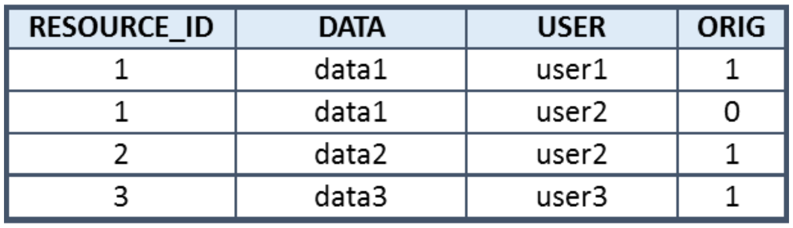
RESOURCE_ID	DATA	USER
1	data1	user1
1	data1	user2
2	data2	user2
3	data3	user3

Figure 6. Data duplication illustration

Internal users

This solution requires additional logic if it should support internal users with access to all data. Since this alternative duplicates data it will also have to hide the duplicates from the internal users. The join in the above picture can be extended to also add a column to indicate if this is the original row, or a duplicate.

Figure 7 shows a table with an ORIG flag set to one for the original rows and zero for the duplicates:



RESOURCE_ID	DATA	USER	ORIG
1	data1	user1	1
1	data1	user2	0
2	data2	user2	1
3	data3	user3	1

Figure 7. LASR table with ORIG column

The ORIG row is used to filter data for internal users by setting “'ORIG'n = 1” as a permission condition for them.

EVALUATION

The project evaluated the five alternatives across a set of different requirements in order to find the optimal alternative. For this project, the impact on the query performance was probably the most important requirement in order to meet the business requirements without having to have a too large and expensive Visual Analytics platform. Table 2 shows the evaluation of each alternative across all the requirements.

Alternative / Evaluation criteria	**Alternative 1 – External identity**	**Alternative 2 - Metadata groups**	**Alternative 3 –Star Schema**	**Alternative 4 – Denormalized table**	**Alternative 5 – Data duplication**
Coverage of Business requirements	- Resource IDs must be concatenated ◑	- Not possible to cover all rules ◔	+ Covers all business requirements ●	+ Covers all business requirements ●	+ Covers all business requirements ●
Query Performance	- text search in query ◑	- Metadata server is xml based, the number of groups causes performance issues ◔	- Joins all rows in the star schema before filtering which can be slow - Query needs to compare long character strings - risk of running out of space in the user list colums ◔	- Query needs to compare long character strings - risk of running out of space in the user list colums - user more memory than alternative 3 ◑	+ Simple query + good query performance - each row is duplicated multiple times (uses more memory) ◑
Implementation complexity	- Complex logic to concatenate and change IDs + No extra data added to LASR ◔	- very complex update logic of groups and permission conditions - many metadata groups + No extra data added to LASR ◔	+ uses standard functionality + Easy to update as security data is saved only in security table - Complex to update security table ◕	+ Less complexity in metadata server - Complex to update LASR ◑	+ No updates, only append/delete + Less complexity in metadata server - Logic ◕
Stability, automation & Flexibility	- Limitation in number of resources per user ◔	+ Simple to add new LASR tables ◑	- Not possible to use star schema for normal usage - stops working if enough users get access to the same accounts ◔	- Complex to add new LASR tables - stops working if enough users get access to the same accounts ◔	+ Simple logic that is easy to run - Not possible to use star schema (many – many join to dimensions) ◕
Conclusion	A possible solution for less complex systems. It is not ideal for the project due to difficulties in concatenating IDs and a maximum number of IDs that can be saved ◑	A possible alternative for solution with a smaller number of unique resources to keep track of. The number of resources in this project is too large for this solution to work effectively. ◔	This solution would work for the project. It allows for easy updating of access data and the addition of new tables. Joining logic reduces performance and it stops working if enough users get access to the same account ◑	A workable solution for this project. The major issue with this solution is that it stops working when enough users has access to the same account. ◑	This solution is usable by the project. It gives the best performance out of the five tested. Data is duplicated in LASR leading to much larger datasets. ◕

Table 2. Evaluation of alternatives

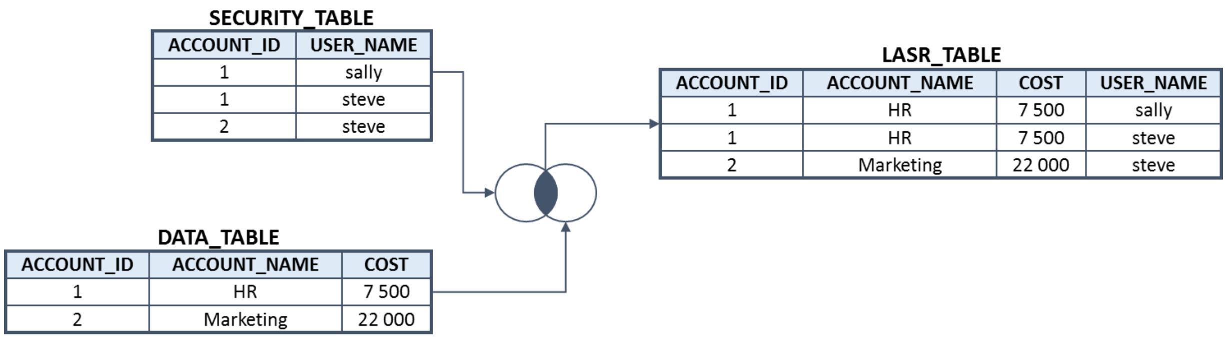
Alternative five was deemed the best for this project. Partly because it showed the best query performance but also because it did not stop working if enough users got access to the same resource.

Another factor is the distribution of users with access to individual accounts. The maximum number of users for an account was not fixed in this project, but it was over 200 at the time of analysis. However, the average number of users with access were less than 10 per account. Alternatives 3 and 4 was also possible to implement, but the USERS column would have to be able to store more than 200 users while mostly storing less than ten. In alternative five, the USER column will always contain one user and can therefore have a small fixed size. As the maximum number of users with access to one account was high, while the average number of users was low the datasets with USERS column took up more space in LASR than the datasets with duplicated rows.

IMPLEMENTATION EXAMPLE

This project selected Alternative 5 – Data duplication for implementation as it covered all business requirements and showed acceptable performance in tests. In the following example, you can see how this was implemented for a report with one table, using account as the base for row-level security.

In the below example, we want to show a cost report for two accounts. Steve has access to both the HR and Marketing accounts while Sally only has access to the HR account. Figure 8 shows the data used in this example.



SECURITY_TABLE

ACCOUNT_ID	USER_NAME
1	sally
1	steve
2	steve

DATA_TABLE

ACCOUNT_ID	ACCOUNT_NAME	COST
1	HR	7 500
2	Marketing	22 000

LASR_TABLE

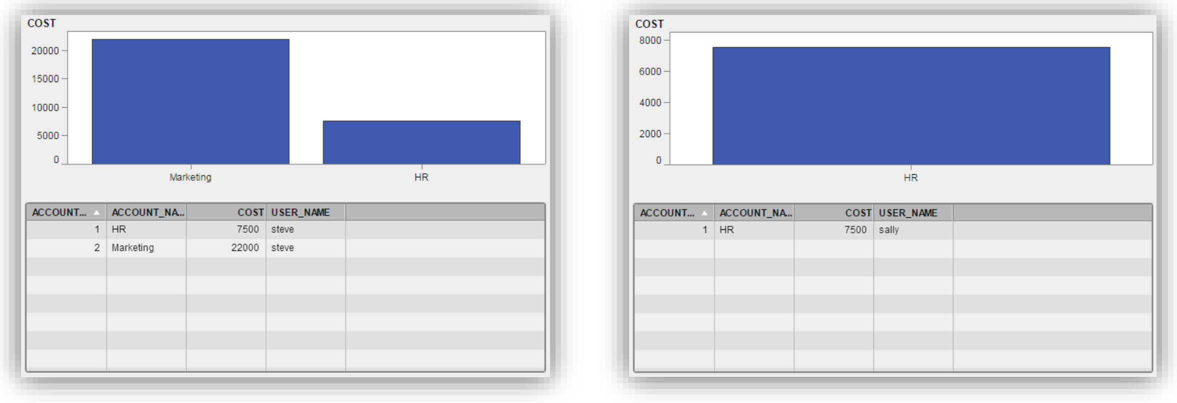
ACCOUNT_ID	ACCOUNT_NAME	COST	USER_NAME
1	HR	7 500	sally
1	HR	7 500	steve
2	Marketing	22 000	steve

Figure 8. Sample data

In this case permission conditions have to be applied to the LASR_TABLE. Filtering should be done on the users' names which allows for an expression like this:

```
'USER_NAME'n = 'SUB::SAS.PersonName'
```

Figure 9 shows the same Visual Analytics report based on the LASR_TABLE, opened by Sally and Steve after the row-level security has been implemented.



ACCOUNT...	ACCOUNT_NA...	COST	USER_NAME
1	HR	7500	steve
2	Marketing	22000	steve

ACCOUNT...	ACCOUNT_NA...	COST	USER_NAME
1	HR	7500	sally

Figure 9. Report viewed by Steve and Sally

UPDATING PERMISSIONS

The major challenge of this project was to satisfy the requirements of reflecting changes in access rights as well as adding or removing users within a few hours of the change. To that end two ETL processes were built to update the LASR tables with changed access rights.

Nightly ETL

To incorporate new data from source databases a Nightly ETL job was used to perform a full join between the security table and the data table, completely refreshing the table in LASR. To minimize downtime in the reports the following steps was used.

Step 1 – Join the tables in SAS work:

```
Proc sql;
   create view work.JOINED_TABLE as
   select
      DATA_TABLE.ACCOUNT_ID,
      DATA_TABLE.ACCOUNT_NAME,
      DATA_TABLE.COST,
      SECURITY_TABLE.USER_NAME
   from
      DATA_TABLE,
      SECURITY_TABLE
   where
      DATA_TABLE.ACCOUNT_ID = SECURITY_TABLE.ACCOUNT_ID;
quit;
```

Step 2 – The created view is uploaded into a temporary compressed LASR table. This allows the reports to be usable while data is uploaded:

```
Data VALIBLA.LASR_DATA_TEMP (squeeze=yes);
   SET JOINED_TABLE;
Run;
```

Step 3 – All rows are removed from the LASR table used by the report:

```
Proc Imstat data=VALIBLA.LASR_DATA;
   deleterows / purge;
Run;
```

Step 4 – Lastly data is copied from the temporary table into the LASR table used by the report and the temporary table is removed:

```
Proc Imstat;
   Table VALIBLA.LASR_DATA;
   Set LASR_DATA_TEMP / Drop;
Run;
```

Intraday ETL

To accommodate the requirements on reflecting changes throughout the day the project created a lightweight ETL that performs delta updates on LASR tables. This intraday ETL is responsible for updating LASR and the Metadata server when new users are added, old users are removed or users access rights are changed. The intraday ETL runs several times throughout the day and only handles the changes since it was ran previously.

The below example code works on the tables ACCOUNTS_TO_ADD and ACCOUNTS_TO_REMOVE that are created in earlier steps of the job. These tables contains the IDs of all the accounts to be added and removed. To comply with security requirements, the ETL will first remove all data associated with accounts where changes has occurred, then re-add the data with the changes applied.

Step 1 – The ETL extracts the subset of the security table for all the accounts that should be added:

```
proc sql;
   create view work.DELTA_SECURITY_TABLE as
   select
      SECURITY_TABLE.ACCOUNT_ID,
      SECURITY_TABLE.USER_NAME
   from
      ACCOUNTS_TO_ADD,
      SECURITY_TABLE
   where
      ACCOUNTS_TO_ADD.ACCOUNT_ID = SECURITY_TABLE.ACCOUNT_ID;
quit;
```

Step 2 – The new delta security table is joined with the data table to create the new rows that should be inserted into the LASR table:

```
proc sql;
   create view work.DATA_TO_ADD as
   select
      DATA_TABLE.ACCOUNT_ID,
      DATA_TABLE.ACCOUNT_NAME,
      DATA_TABLE.COST,
      DELTA_SECURITY_TABLE.USER_NAME
   from
      DATA_TABLE,
      DELTA_SECURITY_TABLE
   where
      DATA_TABLE.ACCOUNT_ID = DELTA_SECURITY_TABLE.ACCOUNT_ID;
quit;
```

Step 3 – The data to be added is uploaded to LASR in a temporary table to make the append action quicker:

```
Data VALIBLA.DATA_TO_ADD;
```

```
     Set WORK.DATA_TO_ADD;
Run;
```

Step 4 – With the data to append prepared the ETL then creates a comma-separated list with the IDs of accounts to be removed. **Note:** care should be taken at this step so the length of the lst variable is not exceeded. In some cases it might be necessary to use multiple variables:

```
Proc Sql;
    Select distinct ACCOUNT_ID into: lst
    separated by ","
    From ACCOUNTS_TO_REMOVE;
Quit;
```

Step 5 – The job now deletes all the accounts in the list from the LASR table used by the report:

```
Proc Imstat data=VALIBLA.LASR_DATA;
    Where ACCOUNT_ID in (&lst);
Run;
    deleterows / purge;
Run;
```

Step 6 – Lastly the data prepared in step 3 is appended to the LASR table and the temporary table is dropped:

```
Proc Imstat;
     Table VALIBLA.LASR_DATA;
     Set DATA_TO_ADD / Drop;
Run;
```

HANDLING MULTIPLE PRODUCT LINES

This project handled data from several product lines. Each product line tracked access rights separately and the data tables never contained data from more than one product line. To differentiate the security models each product line got its own security table. The ETL jobs joined data tables with the security table for the product line from where the data comes from when uploading the data to LASR.

CONCLUSION

This paper reviewed five different ways to implement row-level security based on the requirements and challenges from a real life customer implementation. All the five methods have their strengths and might be the correct method for your challenge. Alternative 5 - data duplication, turned out to be the best method for our implementation project and the paper explains the main parts of how the alternative was implemented.

REFERENCES

SAS Visual Analytics 7.3 Administration Guide, *SAS Documentation,* pp 44-48.

SAS Visual Analytics 7.3 User's Guide, *SAS Documentation,* pp 71-74.

ACKNOWLEDGMENTS

The authors would like to thank the following people for their contributions to the design process and this paper.

- Eric Åtland, SAS Institute Inc.
- Christopher Redpath, SAS Institute Inc.
- David Schwartz, Accenture
- Nikhil Khanolkar, Accenture
- Ismar Slomic, Accenture

RECOMMENDED READING

- *SAS® Visual Analytics User's Guide*
- *SAS® Visual Analytics Administration Guide*
- *SAS® Guide to BI Row-Level Permissions*

CONTACT INFORMATION

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

Levi Lindblom
Organization: Accenture
Phone: +47 906 72 370
E-mail: levi.lindblom@accenture.com

Jonas Lie-Nielsen
Organization: SAS Institute Inc.
Phone: +47 992 16 763
E-mail: Jonas.Lie-Nielsen@sas.com