SAS 639-2017

I Spy PII: Detect, Protect, and Monitor Personal Data with SAS® Data Management

Cecily Hoffritz, SAS Institute A/S (Denmark)

ABSTRACT

The clock is ticking! Is your company ready for May 25, 2018 when the General Data Protection Regulation that affects data privacy laws across Europe comes into force? If companies fail to comply, they incur very large fines and might lose customer trust if sensitive information is compromised. With data streaming in from multiple channels in different formats, sizes, and wavering quality, it is increasingly difficult to keep track of personal data so that you can risk assess it and protect it. SAS® Data Management helps companies on their journey toward governance and compliance involving tasks such as personal data identification, protection, and monitoring. This paper focuses on using SAS® Federation Server and SAS® Data Management Studio from the SAS® Data Management suite to surface and manage that hard-to find-personal data. SAS Federation Server provides you with a universal way to access data in SAS®, Hadoop, Teradata, SQL Server, Oracle, SAP HANA, and other types of data, and the advanced data masking and encryption capabilities of SAS Federation Server can be used when virtualizing data for users. Purpose-built data quality functions are used to perform identification analysis, parsing, matching, and extraction of personal data. You also learn how the exploratory data analysis capability of SAS® Data Management Studio enables you to scan through your data connections to identify and categorize personal data.

INTRODUCTION

In the age of digitalization, protecting personal data and consumer integrity has become a very high priority for the European Union (EU). Personal data is any data related to a living person, and sensitive personal data is about ethnicity, religious belief, sexual preference, medical records, and so on. The General Data Protection Regulation (GDPR) that comes into force in May 2018 provides every EU citizen the right to know how personal data is being used, kept, protected, and deleted. Companies failing to comply with the GDPR might incur fines up to 20 million EUR or 4% of their annual global turnover. The size of the fine and the short time to become GDPR compliant are two motivational factors that are driving companies to review their policies, procedures, and technologies.

SAS Data Management provides numerous capabilities in the areas of data integration and federation, data quality and governance, and it supports the following five-step approach to handle personal data.

1. Access	Access data sources for personal data investigation or business use.
2. Identify	Find, catalog, and analyze personal data attributes, patterns, and contexts to evaluate need for de-identification and risk assessment.
3. Govern	Link systems, processes, and business owners in data flows and ensure integrity of personal data so that it is accurate, complete, and consistent.
4. Protect	Implement data protection safeguards and apply privacy-specific measures such as pseudonymization and anonymization.
5. Audit	Log, monitor, and audit usage of personal data to demonstrate compliance with privacy controls and prove that personal data is not at risk.

Display 1. Five-step Approach to Support Efforts Involving Personal Data

This paper focuses on the strong capabilities of SAS Federation Server to access, identify, protect, and monitor personal data. It also introduces SAS Data Management Studio for ad hoc personal data investigation and SAS® Enterprise Guide to demonstrate the protective measures implemented in SAS

Federation Server. The third step on governance is outside the scope of this paper.

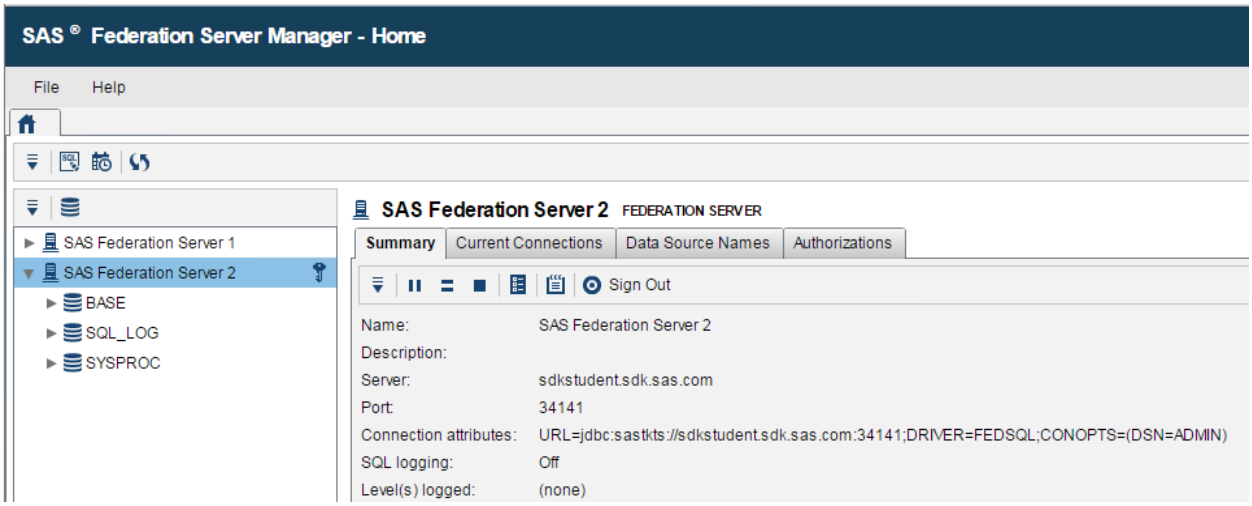
Many companies stipulate that they already know the whereabouts of personal data in their transactional systems, application databases, and other sources, and they already have procedures and policies in place to prove their case. When it comes to analytical data residing in data stores such as warehouses, marts, and lakes, companies might concede that solving business challenges with analytics is top priority, and tracking, tracing, and protecting personal data has not been top of mind.

The GDPR enforcement date of May 2018 is just around the corner, which makes analytical data a prime starting point to implement the above five-step approach. Many customers might prefer to rely on their existing data management processes and tools in their quest to search and protect personal data. If you are required to separate compliance processes from business processes, or you simply want to avoid disrupting current data flows and avoid degrading performance of existing, business critical production jobs, an implementation of SAS Federation Server for personal data investigation and use might prove to be your fastest option to get ready in time.

SAS Federation Server has some very nice key benefits for privacy by design, such as centralized and role-based access to sensitive data, state-of-the-art security, and data masking capabilities including hashing, randomization, and encryption and lastly on-demand data quality to facilitate the identification of personal data if it is not immediately apparent.

ACCESS

With SAS Federation Server, you can implement a data investigation hub, adding those systems that your compliance department has targeted for personal data investigation. SAS Federation Server Manager is the web application that you typically would use for this purpose, and if you find it beneficial not to use the same SAS Federation Server instance that the general populace in your company has access to, you can use a separate SAS Federation Server instance to add those data sources that are up for inspection.

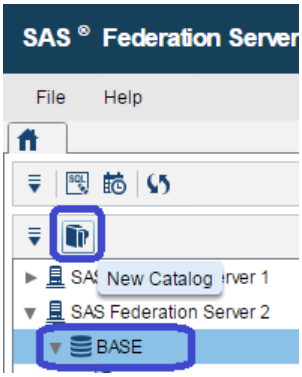


Display 2. Highlighted SAS Federation Server 2 Instance, Ready for Adding Data Sources to the Personal Data Investigation Hub

SCENARIO: SETTING UP ACCESS TO SAS TABLES IN SAS FEDERATION SERVER MANAGER

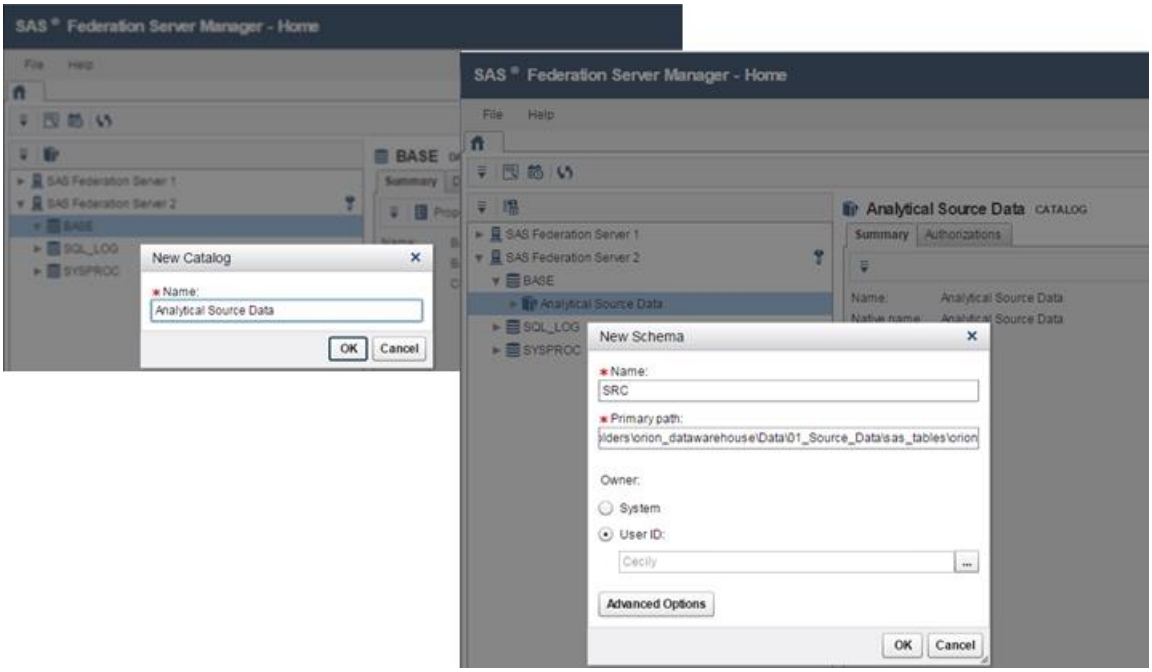
With SAS Federation Server, you can access data from a variety of systems – including SAS, Hadoop, Impala, Netezza, Oracle, SAP HANA, SQL Server, Teradata, and text files. The focus here is showing how easy it is to access your analytical data store that is comprised of SAS tables.

In SAS Federation Server Manager, you highlight the **BASE** data service and click the **New Catalog** icon.



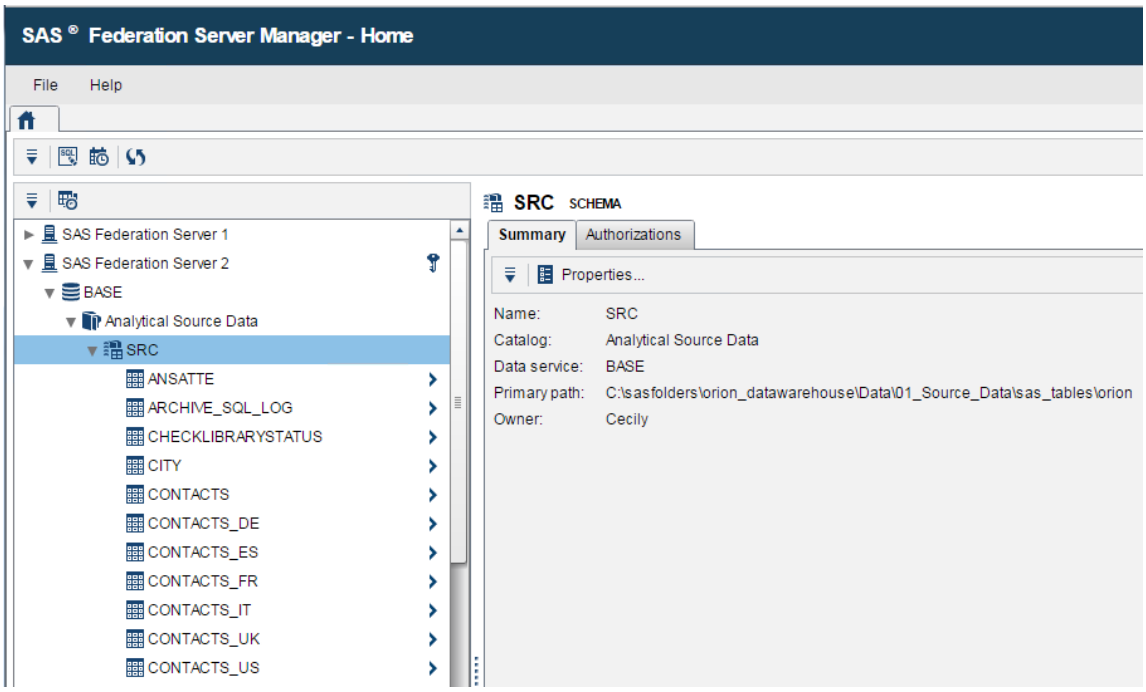
Display 3. Selecting New Catalog

You provide a name for your new catalog, and you provide a name, path, and owner for your new schema.



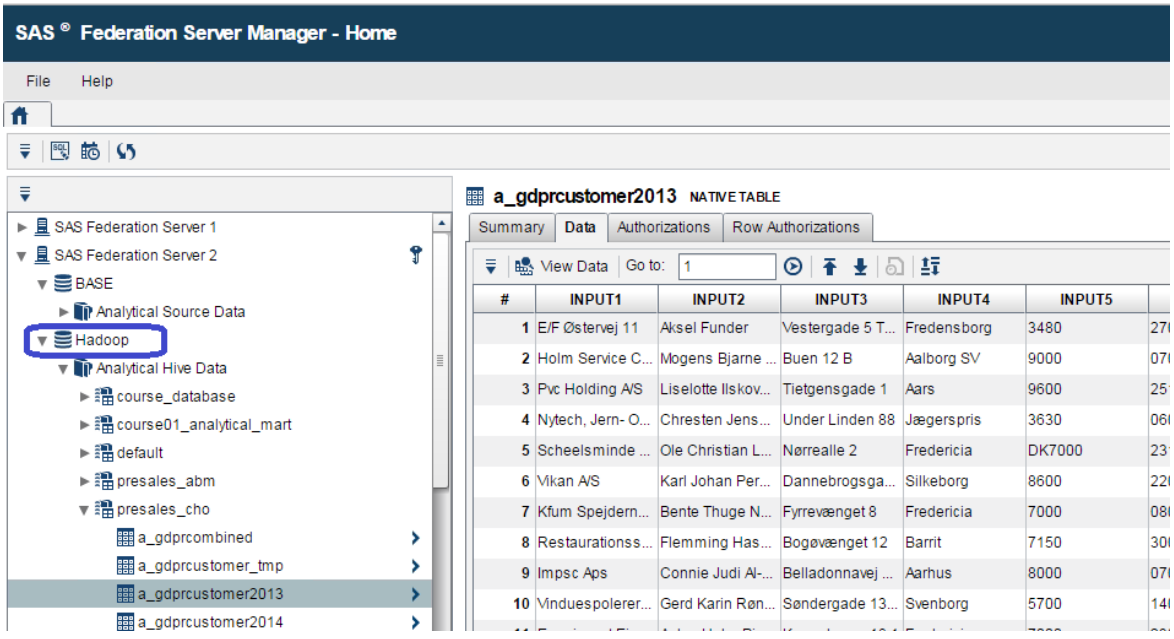
Display 4. New Catalog Selections

Your connection is now ready.



Display 5. Completed Connection

Gaining access to other types of data is not difficult either, and *the SAS® Federation Server Manager 4.2: User's Guide* is very helpful in guiding you to success. Of course, you do need to have specific information ready. For example, when setting up access to a Hive database in Hadoop, you need to know the name of the HIVE server and schema, the port number, and the name of the authentication domain set up on the SAS® Metadata Server. In addition to this, the SAS Federation Server needs to know where the Hadoop configuration files reside.



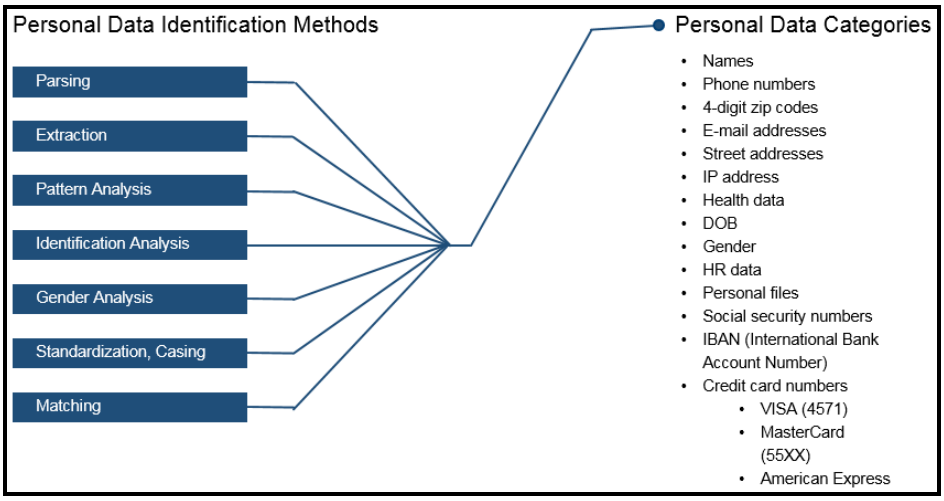
Display 6. Accessing Data in a Hive Database in Hadoop

IDENTIFY

Depending on the quality of your data, identifying and categorizing personal data is probably not without effort. Couple that with programmers having to spend time writing programs with complex logic, your company might find the process of personal data identification daunting and time-consuming.

A potential time-saver is the SAS® Quality Knowledge Base (QKB), a box of prepackaged logic and rules such as identification analysis, standardization, and pattern matching.

The standard QKB supports the management of commonly used contact information for individuals and organizations, such as names, addresses, company names, and phone numbers. A QKB supports locales organized by language and country, for example, English, United States; English, Canada; Danish, Denmark; and Swedish, Sweden. You license each QKB by one or more locales, and if you lack certain logic and rules, you can always customize your QKB. An example of customization is the addition of a special definition to the Danish Locale that helps you detect Danish social security numbers.



Display 7. Overview of Personal Data Identification Methods and Selected Categories in the QKB

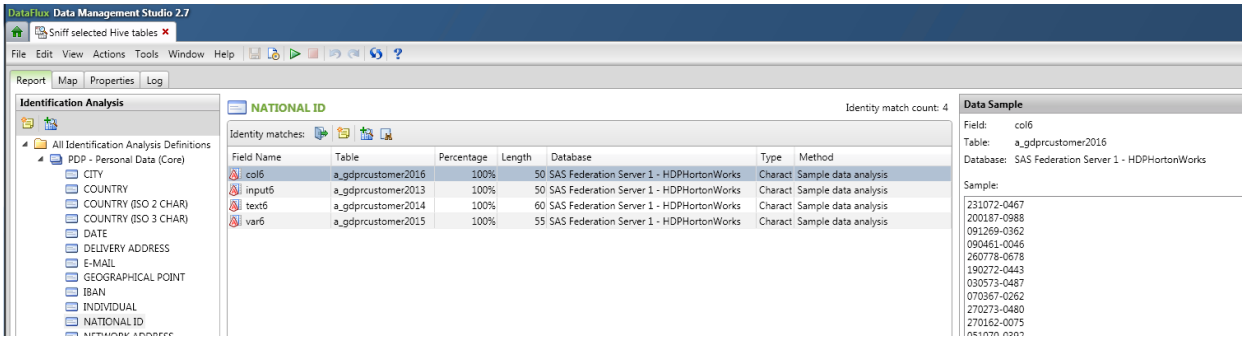
You can use QKB logic to identify personal data from a large variety of SAS Data Management tools such as SAS Enterprise Guide, SAS Display Manager, SAS® Studio, SAS® Data Integration Studio, SAS Data Management Studio, and SAS Federation Server Manager.

This section focuses on three scenarios for personal data identification involving the QKB. The first scenario shows how you can use explorations in SAS Data Management Studio to identify personal data in multiple data sources on an ad hoc basis. The second scenario shows how to locate personal data using FedSQL on a table or view in SAS Federation Server Manager. The third scenario is about using DS2 to scan multiple tables in SAS Federation Server.

SCENARIO 1: IDENTIFYING PERSONAL DATA AD HOC WITH SAS DATA MANAGEMENT STUDIO

If you need to look for personal data in certain data sources here and now, a SAS Data Management Studio exploration is an easy option, and the tool is especially well suited for non-technical compliance personnel, data stewards, or other personnel categories involved in personal data identification.

An exploration provides you with an overview that categorizes your data into personal data categories such as INDIVIDUAL, NATIONAL ID, and IBAN. For example, this overview shows that four columns, col6, input6, text6, and var6 from the NATIONAL ID category have a 100% likelihood to contain Danish social security numbers, and glancing at the data sample in the overview's right pane, you get verification.

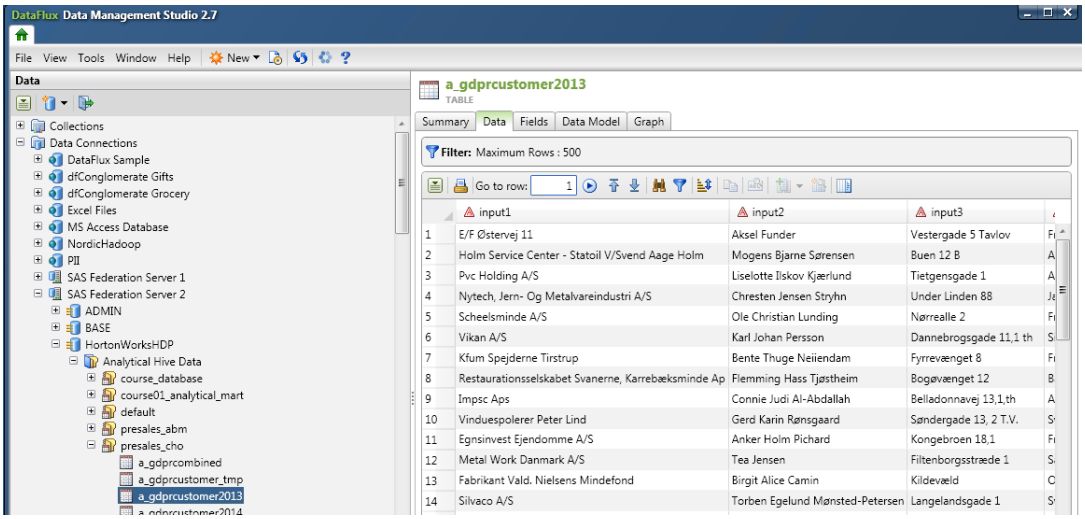


Display 8. Identification Analysis Results Showing 100% Likelihood that Columns Contain Social Security Numbers

Steps to Create an Exploration to Perform Sample Data Analysis

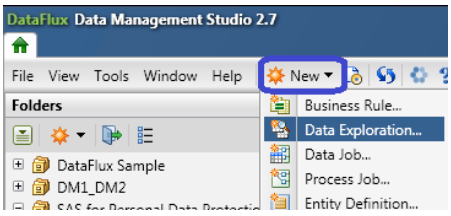
The following steps show you how to create an exploration containing a sample data analysis to find personal data in tables whose columns do not give any indication to content using the Danish locale and a customized QKB definition.

1. In SAS Data Management Studio, single sign on to the SAS Metadata Server, and the data that you defined in SAS Federation Server Manager automatically appears in the **Data** tab.



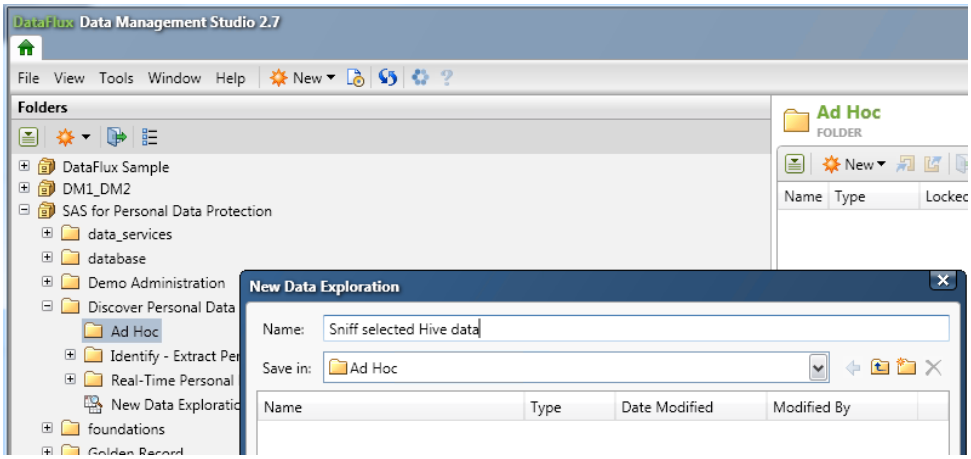
Display 9. Data in the Data Tab

2. In SAS Data Management Studio, create your exploration by selecting **New→Data Exploration** from the menu.



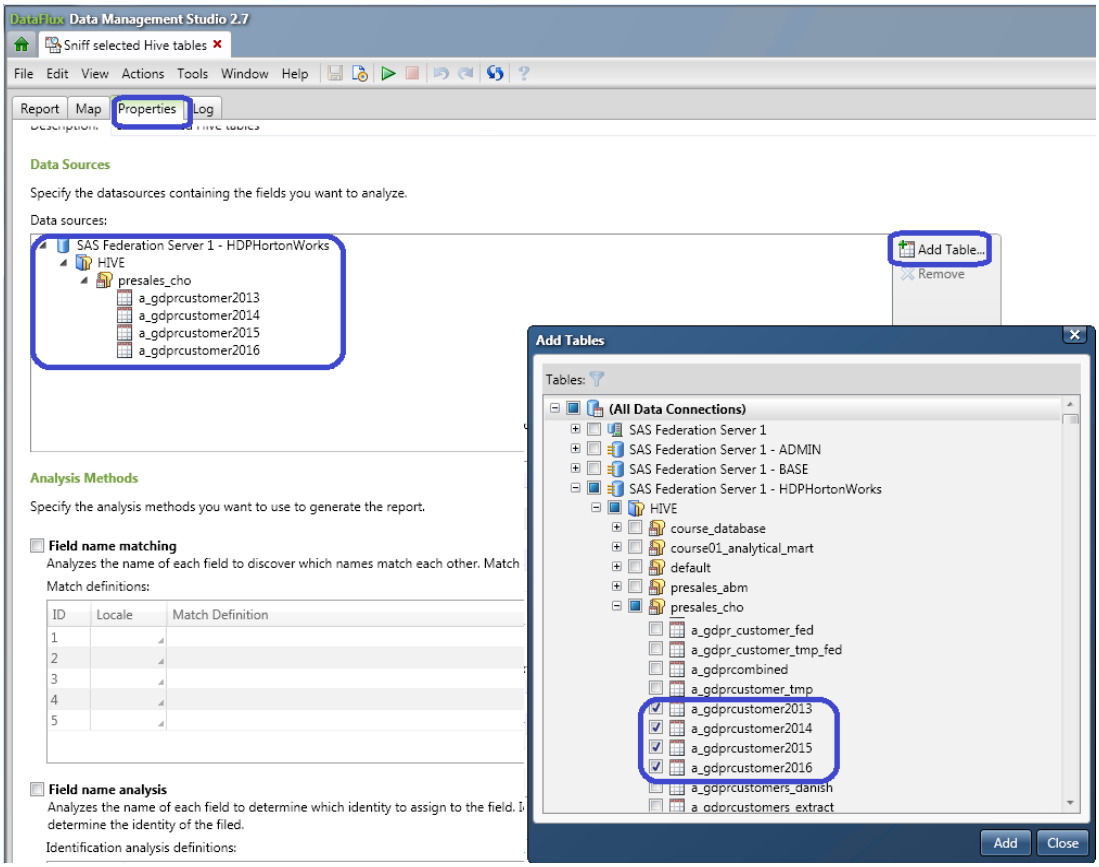
Display 10. New Menu

3. Supply a name and save your exploration in a relevant folder.



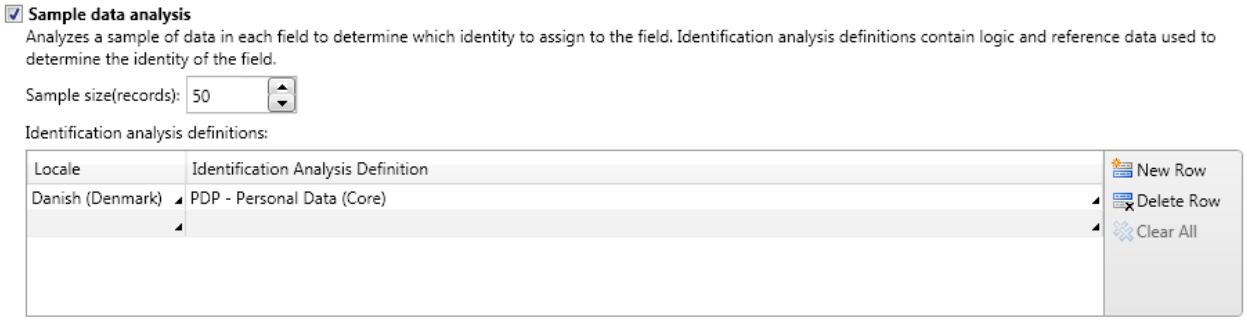
Display 11. Select a Name for Your Exploration

4. In the **Properties** tab, add the tables you want to explore. You can select data at any level. For example, if you want to investigate a whole schema, all you need to do is select the schema.



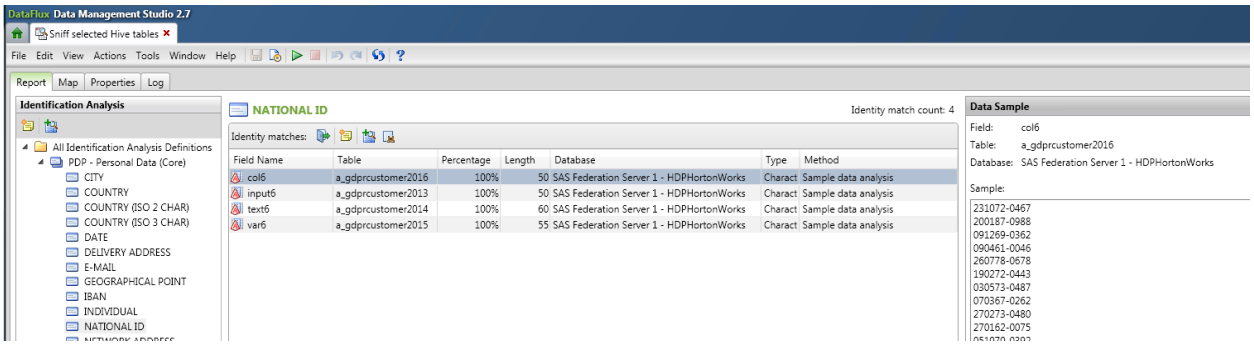
Display 12. Tables to Add to Exploration

5. In the **Properties** tab, perform a sample data analysis where you specify locale and identification analysis definition. *PDP – Personal Data (Core)* is a customized definition that includes identification of Danish social security numbers and other identification categories.



Display 13. Sample Data Analysis

6. Run your exploration. When you run your exploration, you get the previously described overview below.

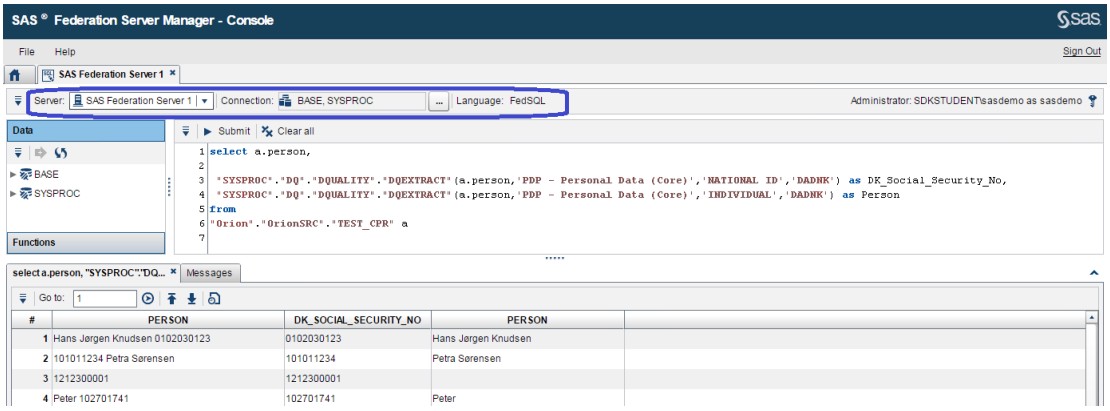


Display 14. Overview

SCENARIO 2: IDENTIFYING PERSONAL DATA WITH FEDSQL IN SAS FEDERATION SERVER MANAGER

In the SQL Console of SAS Federation Server Manager, you can use FedSQL and DQ functions to find personal data in a single table or view that contains a union or a join of multiple tables. The FedSQL program below extracts from a text string the names and content that resembles Danish social security numbers that are in random order. You extract Danish social security numbers and individuals using the DQEXTRACT function, the Danish locale DADNK, the special QKB definition, *PDP – Personal Data (Core),* and the tokens NATIONAL ID and INDIVIDUAL.

When running the code in the SQL console, you select the server, the connections you want to scan, and the language. You also add the connection SYSPROC as it contains the DQ functions, such as DQIDENTIFY, DQPATTERN, DQMATCH, and DQEXTRACT.



Display 15. FEDSQL Program

SCENARIO 3: IDENTIFYING PERSONAL DATA WITH DS2 IN SAS FEDERATION SERVER MANAGER

You can use the DS2 programming language and data quality functions and definitions to scan multiple tables for personal data interactively in the SQL Console in SAS Federation Server Manager, SAS Display Manager, or SQL node in SAS Data Management Studio. You can also run the program in batch mode on a SAS Data Management server or SAS Batch Server, depending on your preference. The example code in the appendix that creates the results below, contains DS2 to scan over all the columns in all the tables of a given catalog and schema. There are several reasons for using DS2. You can run it on SAS Federation Server, which puts the scanning logic closer to the actual data sources. In addition, DS2 is capable of parallel execution, and the work of scanning tables for personal data can be distributed across multiple CPUs and multiple back-end data sources.

Columns with nondescript names that reveal more that 60% likelihood to contain personal data

TABLE NAME	COLUMN NAME	SOCIAL SECURITY NO.	PERSON NAME	CITY	ADDRESS	EMAIL	ORGANIZATION	IBAN	PHONE	PO
A_GDPRCUSTOMER2013	input1	0	1	0	4	0	81	1	0	
	input10	0	0	0	0	0	0	89	0	
	input11	0	11	0	0	0	4	12	9	
	input2	0	94	0	2	0	3	0	0	
	input3	0	1	0	85	0	2	0	0	
	input4	0	4	93	0	0	2	0	0	
	input5	0	0	0	0	0	0	0	0	
	input6	99	0	0	0	0	0	0	0	
	input7	0	0	0	0	0	0	0	96	
	input8	0	0	0	0	0	0	0	0	
	input9	0	0	0	0	98	0	0	0	
	year	0	0	0	0	0	0	0	0	
A_GDPRCUSTOMER2014	text1	0	4	0	3	0	82	2	0	
	text10	0	0	0	0	0	0	91	0	
	text11	0	10	0	0	0	5	10	18	
	text2	0	93	0	3	0	3	0	0	
	text3	0	1	0	87	0	2	0	0	
	text4	0	4	94	0	0	2	0	0	
	text5	0	0	0	0	0	0	0	0	
	text6	99	0	0	0	0	0	0	0	
	text7	0	0	0	0	0	0	0	93	
	text8	0	0	0	0	0	0	0	0	
	text9	0	0	0	0	96	0	0	0	
	year	0	0	0	0	0	0	0	0	
A_GDPRCUSTOMER2015	var1	0	2	1	3	0	79	0	0	
	var10	0	0	0	0	0	0	92	0	
	var11	0	16	0	0	0	6	14	7	
	var2	0	92	0	1	0	3	0	0	

Display 16. A SAS® Visual Analytics Report Showing Results from DS2 Program Submitted in SQL Console of SAS Federation Server Manager

PROTECT

Protection of personal data is about authentication, authorization, and the security auditing and monitoring of users who access personal data. Importantly, protection is also about not compromising the identity of the persons you process and store information about in your growing need for analysis, forecasting, querying, and reporting.

SAS technology has strong security capabilities in general. One of the many good reasons to single out SAS Federation Server is that its security framework provides you with easy to implement methods to mask and encrypt content in SAS tables and all other types of data sources and add column-level and row-level security to them as well. If you have a penchant for using SAS tables for analytical purposes or combining them with data from other systems, SAS Federation Server provides a unified way of dealing with protection of personal data no matter where they come from.

SCENARIO: ADDING SECURITY TO SAS TABLES IN SAS FEDERATION SERVER

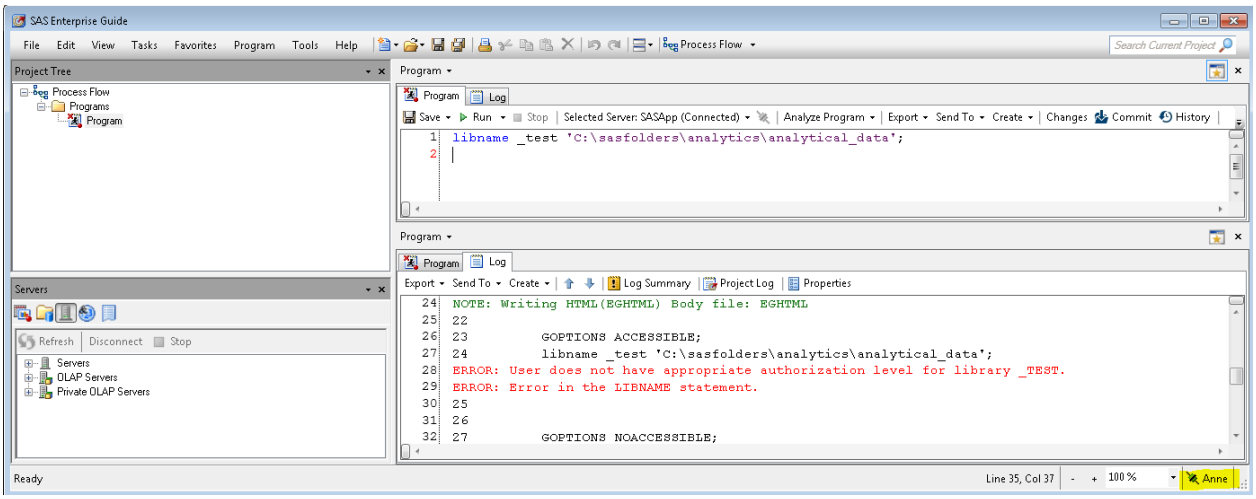
The purpose here is show examples of the implications when restricting access to SAS tables outside a SAS Federation Server context and provide SAS Federation Server authorization to those selected user groups who access the data through SAS Federation Server. In the different examples, there are references to the users Anne, Anita, and Christine. Anne is an analyst and belongs to the Orion Sales Analysts group, Christine can build views, and security prevents Anita accessing data, as she is not in the same group as Anne.

These three steps help you apply the necessary security.

1. Apply operative system security to the directory where SAS tables reside (alternatively apply SAS metadata bound library security to directory containing SAS tables).
2. Create a catalog and a schema in BASE data service. Provide owner of the schema and permissions for the connection.
3. Create a view with FedSQL and add necessary permissions in the view.

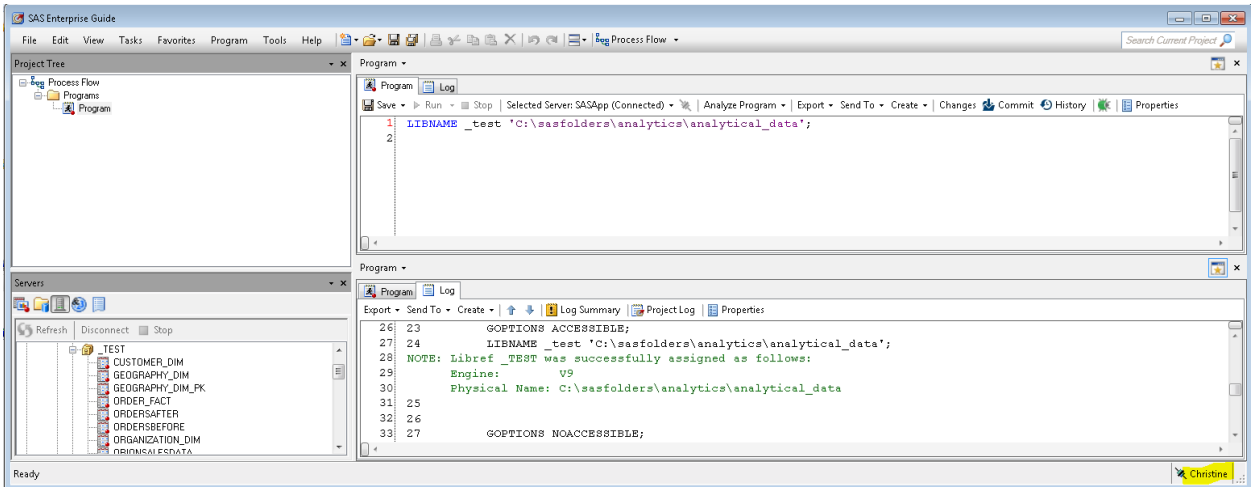
About step 1.

In the screenshot below, the user Anne has submitted a Base SAS engine LIBNAME statement. She is not successful and the log reveals this. The reason is that operative system security does not provide her with physical access to the path in the LIBNAME statement.



Display 17. SAS Enterprise Guide Showing Unsuccessful Library Allocation for Anne

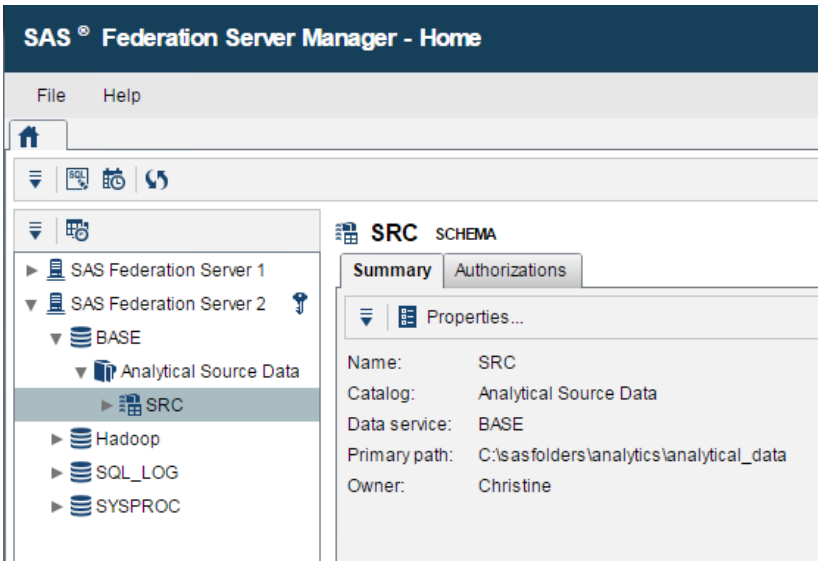
In the following screenshot, you can see that Christine does have physical access to the path because her library assignment is successful. She can also see rows of data in tables.



Display 18. SAS Enterprise Guide Showing Successful Library Allocation for Christine

About step 2.

The SAS Federation Server administrator creates the catalog and schema in the BASE data service and adds Christine as the owner. The SAS metadata administrator provides Christine with the capabilities to use SAS Federation Server Manager through role membership.



Display 19. SAS Federation Server Manager Showing Connection Properties for Schema SRC

About step 3.

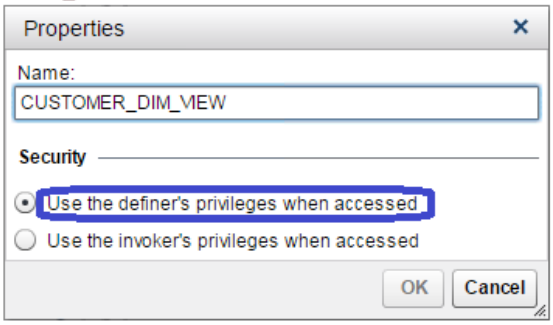
Christine creates a view named CUSTOMER_DIM_VIEW based on this query. It is a definer's rights view, meaning that users will be piggybacking the owner to access the view. Christine also has the option to cache the view to memory and refresh it at scheduled intervals if the underlying system is slow or she wants to provide 24/7 access when the underlying system is down due to maintenance.

CUSTOMER_DIM_VIEW FEDSQL VIEW

Summary | Query | Data | Authorizations | Row Authorizations

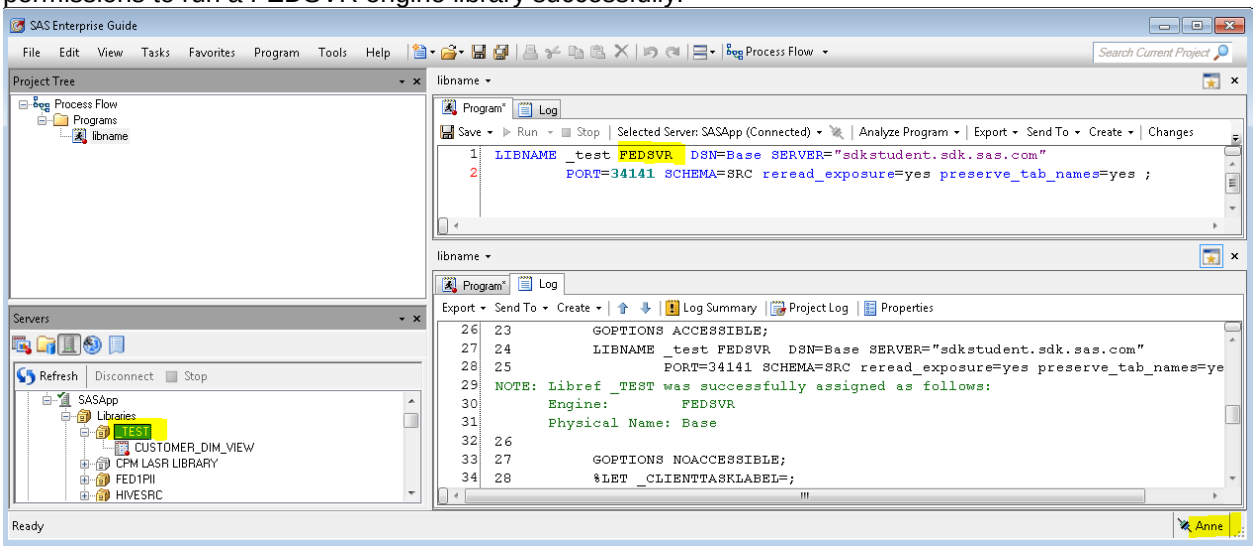
Properties...

```
CREATE VIEW "Analytical Source Data"."SRC"."CUSTOMER_DIM_VIEW" AS
      SELECT
         A."Customer_ID" ,
         A."Customer_Country" ,
         A."Customer_Group" ,
         A."Customer_type" ,
         A."Customer_Gender" ,
         A."Customer_Age" ,
         A."Customer_Age_Group" ,
         A."Customer_Name" ,
         A."Customer_Firstname" ,
         A."Customer_Lastname" ,
         A."Customer_Birth_Date"
      FROM
         "Analytical Source Data"."SRC"."CUSTOMER_DIM" AS A
```



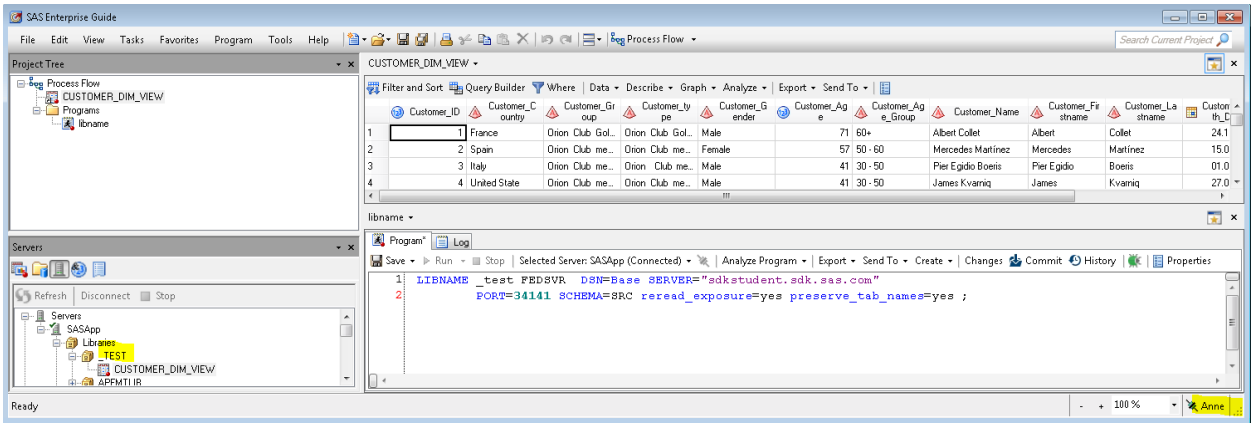
Display 20. The FedSQL Query Code for CUSTOMER_DIM_VIEW and View of Properties

The SAS Federation Server administrator provides the group Orion Sales Analysts, of which Anne is a member, permissions to see Christine's new view, but not the underlying tables in the same path. Recall that Anne was not previously able to run a Base SAS engine library successfully, but now she has permissions to run a FEDSVR engine library successfully.



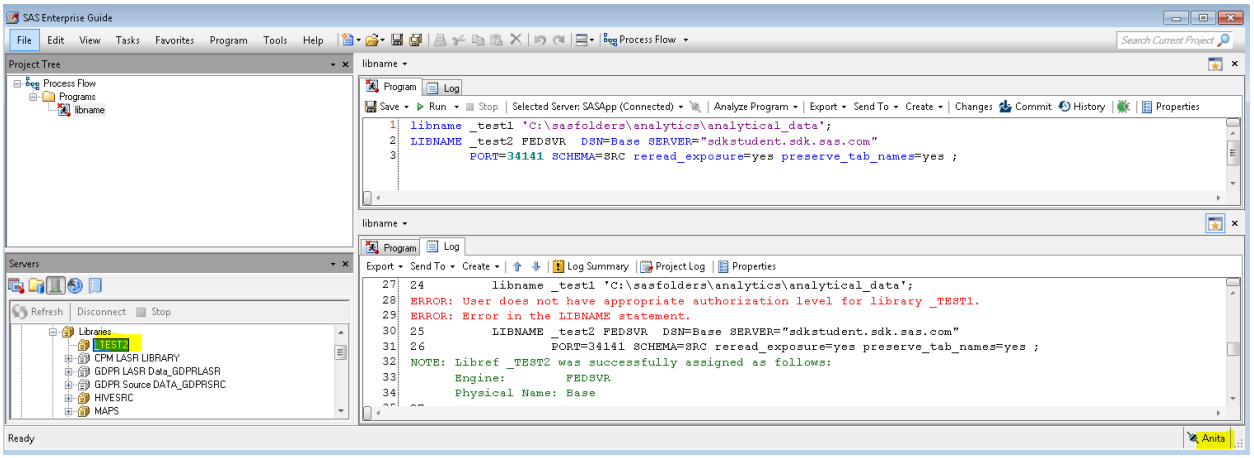
Display 21. SAS Enterprise Guide Showing Library Using FEDSVR Engine and Only the View in the Library

Anne can open the FedSQL view to see the data that is an exact replica of the underlying table Customer_Dim at this point.



Display 22. SAS Enterprise Guide Showing that Anne Can Open the FedSQL View

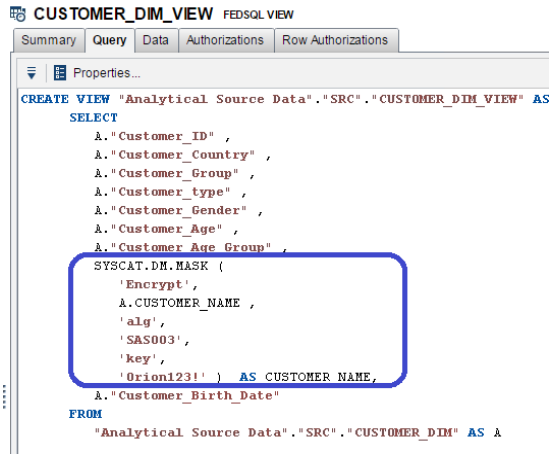
Anita, who is not an Orion Sales Analyst, and does not have physical access to the SAS tables cannot successfully submit a Base SAS engine LIBNAME, but can in fact submit a FEDSVR engine LIBNAME because she has permission to connect to SAS Federation Server. Note in the screenshot below that her library _TEST2 does not show any tables or views, because she does not have permissions granted to see content.



Display 23. SAS Enterprise Guide Showing that Anita Does Not Have Appropriate Permissions

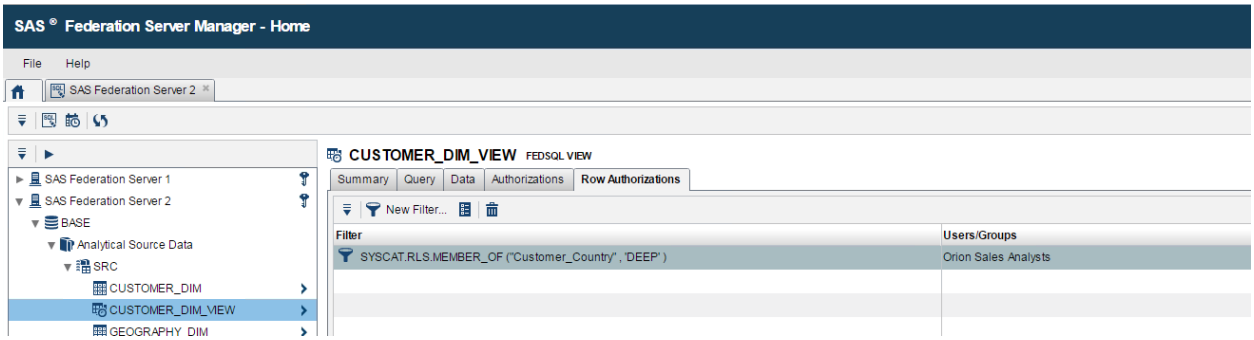
Christine decides to re-create the CUSTOMER_DIM_VIEW with these enhancements.

1. Encryption of the column CUSTOMER_NAME.



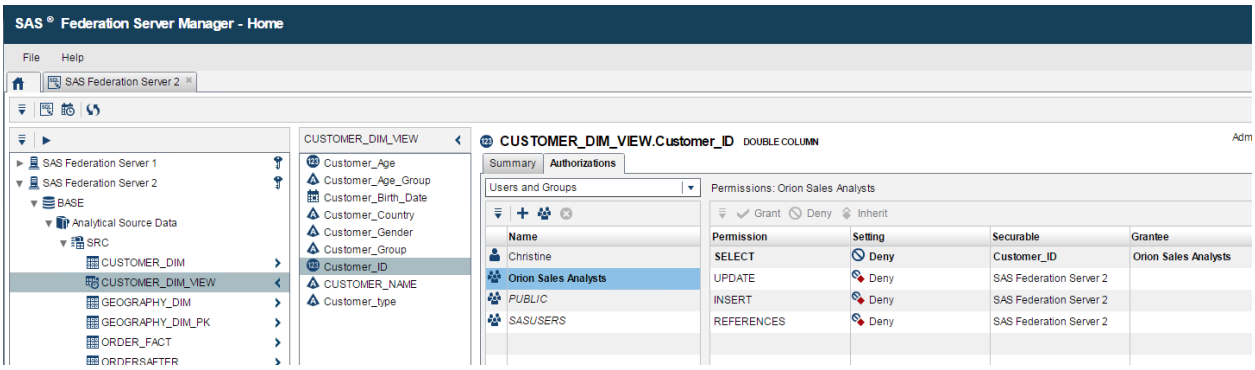
Display 24. Usage of SYSCAT.DM.MASK Function for Encryption

2. Row-level security based on direct or indirect membership of a metadata group called DK that is compared to the value 'DK' in the column Customer_Country.



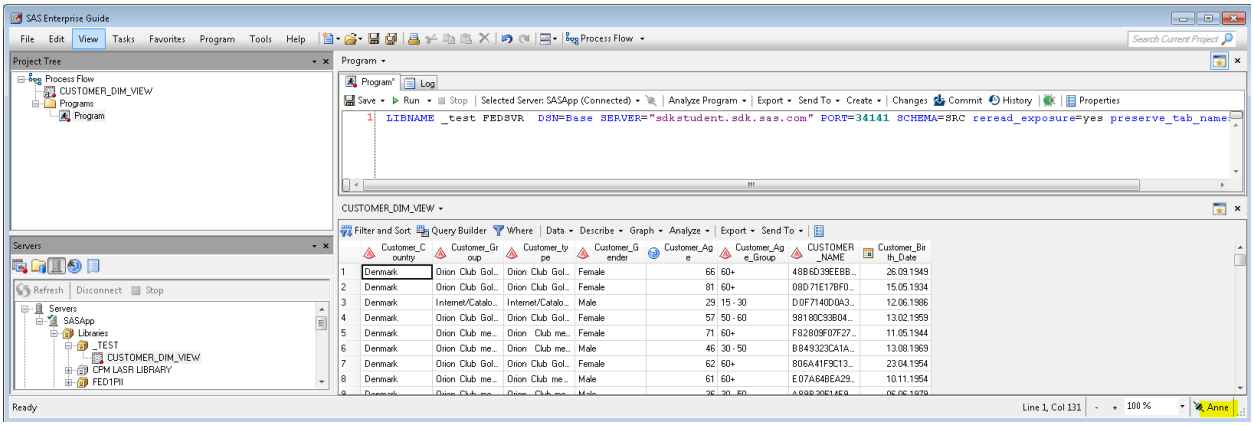
Display 25. Using SYSCAT.RLS.MEMBER_OF Function for Identity-based Filter

3. Hiding the column Customer_ID.



Display 26. Denying Orion Sales Analysts the Ability to See the Column Customer_ID

After Christine has made her changes, Anne will only be able to see data for 'DK'. The column Customer_ID is not visible to her, and she cannot see CUSTOMER_NAME in plain text.



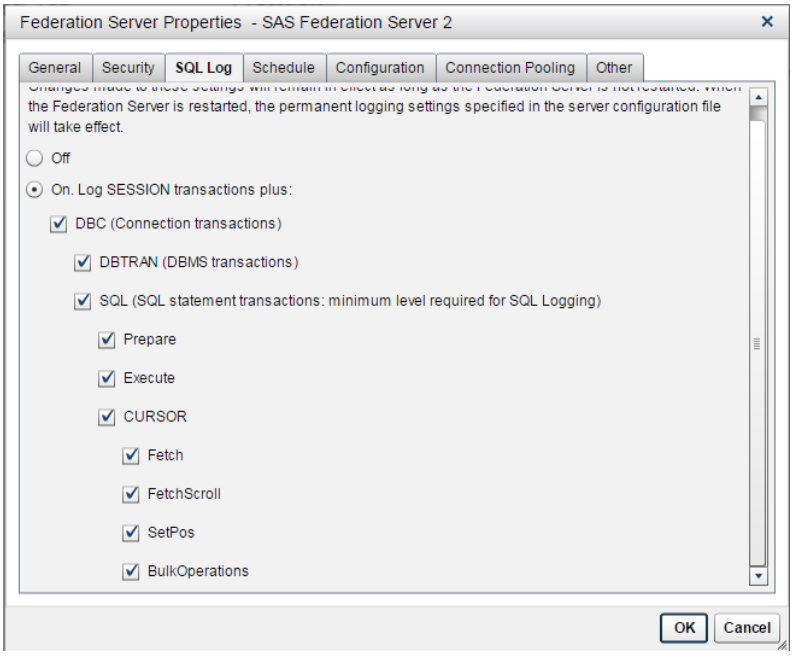
Display 27. Effective Permissions Limiting Anne's Access to Certain Rows and Columns

AUDIT

Auditing is about logging and monitoring the usage of personal data to demonstrate compliance with privacy controls. It is also about analyzing and reporting to prove that personal data is not at risk. This section focuses on SQL logging that provides the ability to view SQL statements submitted to SAS Federation Server. SQL statements can be combined with other information, for example, the user ID of the user who submitted the SQL, and information about prepare, execute, and cursor phases. Metrics are also available, including elapsed time, number of rows fetched, and the size of data fetched or inserted.

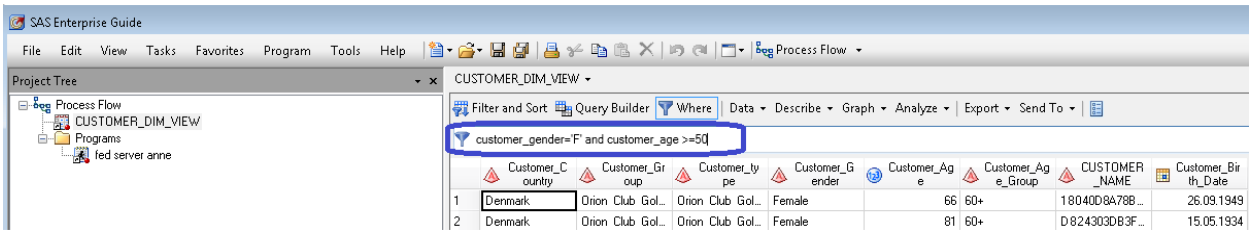
SCENARIO: ACTIVATING SQL LOG TO MONITOR USAGE OF DATA

You activate SAS Federation Server logs in configuration files or interactively in SAS Federation Server Manager, in the **SQL Log** tab in Properties for the requested Federation Server instance.

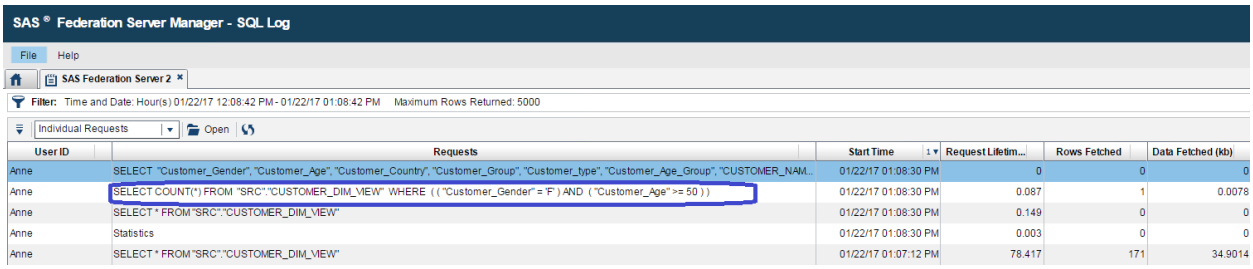


Display 28. Activating the SQL Log in SAS Federation Server Manager

In SAS Enterprise Guide, when Anne opens data from the SAS Federation Server hub and submits a WHERE clause, her activity is automatically registered in the SQL log, making it easy to analyze her usage of personal data.



Display 29. Submitting an Interactive WHERE Clause in SAS Enterprise Guide



Display 30. Interactive WHERE Clause Showing Up in the SAS Federation Server SQL Log

CONCLUSION

With SAS Data Management tools, your company has the opportunity to provide the technological support to identify, govern, protect, and monitor personal data. This paper focused on SAS Federation Server, providing you with the necessary insight on important capabilities so that you might contemplate adding SAS Federation Server to your data management environment for personal data investigation or as your new analytical data hub. Likewise, this paper has proven to you that SAS Data Management Studio is worth looking into because of its strong capabilities for exploration, profiling, and data quality.

APPENDIX

```
/*********************************************************************/
/* Developers: Brian Hess, Tom Cole, SAS Institute Inc.              */
/*                                                                   */
/* This program utilizes DS2 and DQ functions to identify personal   */
/* data for DSNs defined in Federation Server. This program uses     */
/* the QKB definition PDP - Personal Data (Core) that is found for a */
/* number of locales. It contains a large list of personal data      */
/* categories. Contact your SAS representative to hear more about the */
/* custom definition PDP - Personal Data (Core).                     */
/*                                                                   */
/* For your initial test, run this program in Federation Server      */
/* Manager's SQL Console where you connect to relevant Federation    */
/* Server instance, add DSNs you want to search, add SYSPROC DSN     */
```

```
/* and select the language DS2.                                          */
/*                                                                       */
/* Edit the program below to:                                            */
/*   - set the SAMPLE size (default value is 100)                        */
/*   - select a subset of the data                                       */
/*   - set the number of threads (default is threads=4)                  */
/*   - specify catalog, schema and table for results                     */
/* To edit, search the text EDIT HERE in the program.                    */
/*                                                                       */
/* After initial test, submit the edited program to the Fed Server       */
/* via any of the following:                                             */
/*   - SQL Console                                                       */
/*   - a JDBC or ODBC connection (send the program as the prepared       */
/*       statement)                                                      */
/*   - from a SQL Query node in DM Studio/Server                         */
/*   - PROC DS2 using the NOPROMPT option.                               */
/*       Note:  for this case, the program needs slight changes to       */
/*       adopt the PROC syntax.                                          */
/*************************************************************************/

/*************************************************************************/
/* SECTION A: helper packages to implement linked lists                  */
/*************************************************************************/
/*-- define a simple list element --*/
package myElement/overwrite = yes;
  dcl package myElement next;    /* single linked list */
  declare nchar(512) c;          /* character value    */
  declare double     d;          /* double value       */

  method myElement(nchar(512) c);
    this.c = c;
  end;
  method myElement(double d, int f);
    this.d = d;
  end;
endpackage;

/*-- define a linked list of elements --*/
package myList/overwrite=yes;
  dcl package myElement front;
```

```
  dcl package myElement back;

  method add(package myElement element);
    if( null( front ) ) then do;
      front = element;
      back = element;
      return;
    end;
    else do;
      back.set_next(element);
      back = element;
    end;
  end;

  /*-- add a custom delete() method, called automatically --*/
  method delete();
    dcl package myElement cur next;
    cur = front;
    do while (^null( cur ) );
      next = cur.next;
      cur.delete();
      cur = next;
    end;
  end;
endpackage;
/*********************************************************************/
/* Section B: The worker thread, which scans for all tables and      */
/*            columns, and uses DQ methods to assess the data in each */
/*            of those columns.                                      */
/*********************************************************************/
thread tpgm / overwrite=yes;
  declare package sqlstmt stmt_identification;
  declare package sqlstmt stmt_columns;
  declare double start_dt;
  declare double end_dt;
  declare nchar(20) start_fmt;
  declare nchar(20) end_fmt;
  declare nchar(512) TABLE_CAT;
  declare nchar(512) TABLE_SCHEM;
  declare nchar(512) TABLE_NAME;
```

```
  declare nchar(512) COLUMN_NAME;
  declare double DATA_TYPE;
  declare nchar(512) TYPE_NAME;
  declare double COLUMN_SIZE;
  declare double BUFFER_LENGTH;
  declare integer DECIMAL_DIGITS;
  declare integer NUM_PREC_RADIX;
  declare integer NULLABLE;
  declare char(1024) REMARKS;
  declare char(256) COLUMN_DEF;
  declare integer TKTS_DATA_TYPE;
  declare integer TKTS_DATETIME_SUB;
  declare integer CHAR_OCTET_LENGTH;
  declare integer ORDINAL_POSITION;
  declare char(5) IS_NULLABLE;
  declare char(20) IDENTIFICATION;
  declare char(16384) dq_identify;
  declare char(1024) dq_columns;
  declare nchar(1000) error_text;
  declare integer threadid;
  declare integer rc;
  declare integer sample_size;
  declare integer total_rows;

/*********************************************************************/
/* In this version, there is an implicit limit of 1000 columns       */
/* per table for storage of statistics. This could be implemented    */
/* as a HASH object instead of as array to provide greater           */
/* flexibility.                                                      */
/*********************************************************************/
  declare double  total_email     [ 1000 ];
  declare double  total_iban      [ 1000 ];
  declare double  total_network   [ 1000 ];
  declare double  total_pay       [ 1000 ];
  declare double  total_url       [ 1000 ];
  declare double  total_cntry     [ 1000 ];
  declare double  total_cntry2    [ 1000 ];
  declare double  total_cntry3    [ 1000 ];
  declare double  total_date      [ 1000 ];
  declare double  total_geo       [ 1000 ];
```

```
   declare double  total_indiv     [ 1000 ];
   declare double  total_org       [ 1000 ];
   declare double  total_deladdr   [ 1000 ];
   declare double  total_city      [ 1000 ];
   declare double  total_postal    [ 1000 ];
   declare double  total_phone     [ 1000 ];
   declare double  total_unkn      [ 1000 ];
   declare double  total_natid     [ 1000 ];
   declare double  total_vehreg    [ 1000 ];
   declare double  total_blnk      [ 1000 ];
    declare double  total_nonchar   [ 1000 ];
   declare double  email_pct;
   declare double  iban_pct;
   declare double  network_pct;
   declare double  pay_pct;
   declare double  url_pct;
   declare double  cntry_pct;
   declare double  cntry2_pct;
   declare double  cntry3_pct;
   declare double  date_pct;
   declare double  geo_pct;
   declare double  indiv_pct;
   declare double  org_pct;
   declare double  deladdr_pct;
   declare double  city_pct;
   declare double  postal_pct;
   declare double  phone_pct;
   declare double  unkn_pct;
   declare double  natid_pct;
   declare double  vehreg_pct;
   declare double  blnk_pct;
declare integer column_ord;
declare package myElement column_element;
declare package myElement data_type_element;
declare package myElement type_element;
declare package myElement column_size_element;
declare package myElement buffer_length_element;
declare package myList column_list;
declare package myList data_type_list;
declare package myList type_list;
```

```
declare package myList column_size_list;
declare package myList buffer_length_list;

keep start_fmt end_fmt TABLE_CAT TABLE_SCHEM TABLE_NAME COLUMN_NAME
     DATA_TYPE TYPE_NAME COLUMN_SIZE BUFFER_LENGTH total_rows
     email_pct iban_pct network_pct pay_pct url_pct cntry_pct cntry2_pct
     cntry3_pct date_pct geo_pct indiv_pct org_pct deladdr_pct
     city_pct postal_pct phone_pct unkn_pct natid_pct vehreg_pct blnk_pct
     error_text threadid;
method run();
  declare nchar(20) value;
  /*****************************************************************/
     /* EDIT HERE EDIT HERE EDIT HERE EDIT HERE EDIT HERE EDIT HERE    */
     /* Specify your sample size - optional                            */
     /*****************************************************************/
  sample_size = 100;
  /*****************************************************************/
  /* EDIT HERE EDIT HERE EDIT HERE EDIT HERE EDIT HERE EDIT HERE    */
  /* REQUIRED                                                       */
  /* NOTE: To customize for specific schema, etc. you can add the    */
  /* WHERE clause to the set statement below. For example,           */
  /*                                                                */
  /* set { ...  where (TABLE_CAT = 'MYCATALOG')                     */
  /*              and (TABLE_SCHEM='MYSCHEMA') };                   */
   /*                                                               */
   /* NOTE: Some data connections require that you filter by adding  */
   /* an IF statement below the set statement. For example,          */
   /*                                                               */
   /* if not ((TABLE_CAT ='MYCATALOG' and TABLE_SCHEM = 'MYSCHEMA')) */
  /* then goto next_table_loop;                                      */
   /*                                                               */
  /*****************************************************************/
  set { select RTRIM(TABLE_CAT)   || '.' ||
              RTRIM(TABLE_SCHEM) || '.' ||
              RTRIM(TABLE_NAME) as NAME,
              TABLE_CAT, TABLE_SCHEM, TABLE_NAME, REMARKS
              from dictionary.tables
   /* Example filter for SQL Server                                 */
   /*     where (TABLE_CAT = 'PIIData') and (TABLE_SCHEM='dbo')      */
```

```
      /* Example filter for SAS tables                                          */
      /*     where (TABLE_CAT = 'Orion') and (TABLE_SCHEM='investigation_data')  */
         };
      by NAME;

/* Example filter for HIVE                                                     */
/*   if not ( (TABLE_CAT ='HIVE' and TABLE_SCHEM = 'presales_cho') ) then */
/*   goto next_table_loop;   */

      if( (TABLE_CAT = 'DICTIONARY') or (TABLE_CAT = 'SYSPROC') ) then
        goto next_table_loop;

      /*****************************************************************/
      /* NOTE: To customize for specific tables, add code like this:    */
      /* if (                                                          */
      /*   TABLE_NAME = 'abc' or                                       */
      /*   TABLE_NAME = 'TESTTABLE'                                    */
      /* ) then ;                                                      */
      /* else goto next_table_loop;                                    */
      /*****************************************************************/
      if( REMARKS = 'FEDSQL VIEW' ) then
        goto next_table_loop;
      start_dt = datetime();

  /*********************************************************************/
  /* SECTION C: Construct a SQL query to access the column information  */
  /*            for each table                                         */
  /*********************************************************************/
        email_pct       =           0;
        iban_pct        =           0;
        network_pct     =           0;
        pay_pct         =           0;
        url_pct         =           0;
        cntry_pct       =           0;
        cntry2_pct      =           0;
        cntry3_pct      =           0;
        date_pct        =           0;
        geo_pct         =           0;
        indiv_pct       =           0;
```

```
   org_pct         =            0;
   deladdr_pct     =            0;
   city_pct        =            0;
   postal_pct      =            0;
   phone_pct       =            0;
   unkn_pct        =            0;
   natid_pct       =            0;
   vehreg_pct      =            0;
   blnk_pct        =            0;
   total_rows      =            0;
   threadid        = _threadid_;
   rc              =            0;
   column_list        = _new_ myList();
   data_type_list     = _new_ myList();
   type_list          = _new_ myList();
   column_size_list   = _new_ myList();
   buffer_length_list = _new_ myList();

dq_columns = 'select '                                          ||
             '  COLUMN_NAME, TYPE_NAME, DATA_TYPE, '            ||
             '  COLUMN_SIZE, BUFFER_LENGTH '                    ||
             'from dictionary.columns '                         ||
             'where (TABLE_CAT='''  || TRIM(TABLE_CAT)   || ''')' ||
             '  and (TABLE_SCHEM='''|| TRIM(TABLE_SCHEM) || ''')' ||
             '  and (TABLE_NAME=''' || TRIM(TABLE_NAME)  || ''')' ;

stmt_columns = _new_ sqlstmt();

rc = stmt_columns.Prepare(dq_columns);
if( rc ne 0 ) then do;
  error_text = 'Could not prepare:  ' || dq_columns;
  COLUMN_NAME = '<ALL>';
  goto next_table;
  end;

rc = stmt_columns.execute();
if( rc ne 0 ) then do;
  error_text = 'Could not execute:  ' || dq_columns;
  COLUMN_NAME = '<ALL>';
  goto next_table;
```

```
    end;

    rc = stmt_columns.bindResults([COLUMN_NAME TYPE_NAME DATA_TYPE
                                   COLUMN_SIZE BUFFER_LENGTH]);
    if( rc ne 0 ) then do;
      error_text = 'Could not bind columns:  ' || dq_columns;
      COLUMN_NAME = '<ALL>';
      goto next_table;
    end;

/*********************************************************************/
/* SECTION D: Build a new SELECT statement for the desired columns in */
/*            this table                                             */
/*********************************************************************/
    dq_identify = 'SELECT';
    column_ord=1;

    /* Loop over all columns, building the Select statement */
    do while (rc = 0);
      rc = stmt_columns.fetch();
      if( rc ne 0 ) then do;
        continue;
      end;

      /* Capture column-specific info, per ordinal */
      column_element = _new_ myElement(COLUMN_NAME);
      column_list.add(column_element);

      data_type_element = _new_ myElement(DATA_TYPE, 0);
      data_type_list.add(data_type_element);

      type_element = _new_ myElement(TYPE_NAME);
      type_list.add(type_element);

      column_size_element = _new_ myElement(COLUMN_SIZE, 0);
      column_size_list.add(column_size_element);

      buffer_length_element = _new_ myElement(BUFFER_LENGTH, 0);
      buffer_length_list.add(buffer_length_element);
```

```
  /*************************************************************/
  /* Build SELECT statement with DQIDENTIFY function call for  */
  /* column, resulting in a query that looks like:             */
  /*                                                           */
  /*    SELECT SYSPROC.DQ.DQUALITY.DQIDENTIFY( [COLUMN_NAME],  */
  /*                              'PDP - Personal Data (Core)',*/
  /*                                          'DADNK' )        */
  /*           AS [COLUMN_NAME],                               */
  /*           ...                                             */
  /*    FROM [TABLE_CAT].[TABLE_SCHEM].[TABLE_NAME]            */
  /*    LIMIT [SAMPLE_SIZE]                                    */
  /*************************************************************/
  if( column_ord > 1 ) then
     dq_identify = TRIM(dq_identify) || ',';

  /*************************************************************/
  /* See if the DATA_TYPE is one of the many character data types */
  /* we support. If it isnt, create a constant value showing a */
  /* skipped column, else add a call to the DQIDENTIFY method.  */
  /*************************************************************/
  if( (DATA_TYPE ne  1) and (DATA_TYPE ne 12 ) and
      (DATA_TYPE ne -1) and (DATA_TYPE ne -8 ) and
      (DATA_TYPE ne -9) and (DATA_TYPE ne -10)
    ) then
     dq_identify = TRIM(dq_identify) || ' ''SKIP_NONCHAR'' AS "' ||
                   TRIM(COLUMN_NAME) || '"';
  else
     dq_identify = TRIM(dq_identify)                          ||
                   ' CAST( SYSPROC.DQ.DQUALITY.DQIDENTIFY( "'  ||
                   TRIM(COLUMN_NAME)                          ||
           '",''PDP - Personal Data (Core)'',''DADNK'') AS CHAR(20)) '  ||
                   ' AS "' || TRIM(COLUMN_NAME) || '"';

  /* Go to next column ordinal */
  column_ord = column_ord + 1;

end; /* for all columns in the table */

/* If we cannot get columns for the table, then skip it */
if( column_ord = 1 ) then do;
```

```
      error_text = 'No columns found for table:  "' ||
                   TRIM(TABLE_CAT)   || '"."' ||
                   TRIM(TABLE_SCHEM) || '"."' ||
                   TRIM(TABLE_NAME)  || '"';
      COLUMN_NAME = '<ALL>';
      rc = 1;
      goto next_table;
    end;

    dq_identify = TRIM(dq_identify) || ' FROM "'  ||
                  TRIM(TABLE_CAT)   || '"."'      ||
                  TRIM(TABLE_SCHEM) || '"."'      ||
                  TRIM(TABLE_NAME)  || '" LIMIT ' || sample_size;


/*********************************************************************/
/* SECTION E: Execute the query to read the sample from the target    */
/*            table.                                                 */
/*********************************************************************/
    stmt_identification = _new_ sqlstmt();
    rc = stmt_identification.Prepare(dq_identify);
    if( rc ne 0 ) then do;
      error_text = 'Could not prepare:  ' || dq_identify;
      COLUMN_NAME = '<ALL>';
      goto cleanup;
      end;

    rc = stmt_identification.execute();
    if( rc ne 0 ) then do;
      error_text = 'Could not execute:  ' || dq_identify;
      COLUMN_NAME = '<ALL>';
      goto cleanup;
    end;

    total_rows = 0;
    do while (rc = 0);
      rc = stmt_identification.fetch();
      if( rc ne 0 ) then do;
         continue;
      end;
```

```
column_ord=1;
do while (column_ord <= stmt_identification.GETCOLUMNCOUNT());

  if( total_rows = 0 ) then do;
    /* Reset all stats */

            total_email[ column_ord ]         = 0;
            total_iban[ column_ord ]          = 0;
            total_network[ column_ord ]     = 0;
            total_pay[ column_ord ]           = 0;
            total_url[ column_ord ]           = 0;
            total_cntry[ column_ord ]         = 0;
            total_cntry2[ column_ord ]        = 0;
            total_cntry3[ column_ord ]        = 0;
            total_date[ column_ord ]          = 0;
            total_geo[ column_ord ]           = 0;
            total_indiv[ column_ord ]         = 0;
            total_org[ column_ord ]           = 0;
            total_deladdr[ column_ord ]     = 0;
            total_city[ column_ord ]          = 0;
            total_postal[ column_ord ]        = 0;
            total_phone[ column_ord ]         = 0;
            total_unkn[ column_ord ]          = 0;
            total_natid[ column_ord ]         = 0;
            total_vehreg[ column_ord ]        = 0;
            total_blnk[ column_ord ]          = 0;
            total_nonchar[ column_ord ]     = 0;
       end;

  value = stmt_identification.GETNCHAR(column_ord);
  if( value = 'SKIP_NONCHAR               ' ) then
    total_nonchar[ column_ord ] = total_nonchar[ column_ord ] + 1;
       else if ( value = 'E-MAIL              ' ) then
    total_email[ column_ord ] = total_email[ column_ord ] + 1;
       else if ( value = 'IBAN                ' ) then
    total_iban[ column_ord ] = total_iban[ column_ord ] + 1;
       else if ( value = 'NETWORK ADDRESS     ' ) then
    total_network[ column_ord ] = total_network[ column_ord ] + 1;
       else if ( value = 'PAYMENT CARD NUMBER ' ) then
```

```
      total_pay[ column_ord ] = total_pay[ column_ord ] + 1;
         else if ( value = 'URL                ' ) then
      total_url[ column_ord ] = total_url[ column_ord ] + 1;
         else if ( value = 'COUNTRY            ' ) then
      total_cntry[ column_ord ] = total_cntry[ column_ord ] + 1;
         else if ( value = 'COUNTRY (ISO 2 CHAR)' ) then
      total_cntry2[ column_ord ] = total_cntry2[ column_ord ] + 1;
         else if ( value = 'COUNTRY (ISO 3 CHAR)' ) then
      total_cntry3[ column_ord ] = total_cntry3[ column_ord ] + 1;
         else if ( value = 'DATE               ' ) then
      total_date[ column_ord ] = total_date[ column_ord ] + 1;
         else if ( value = 'GEOGRAPHICAL POINT  ' ) then
      total_geo[ column_ord ] = total_geo[ column_ord ] + 1;
         else if ( value = 'INDIVIDUAL         ' ) then
      total_indiv[ column_ord ] = total_indiv[ column_ord ] + 1;
         else if ( value = 'ORGANIZATION       ' ) then
      total_org[ column_ord ] = total_org[ column_ord ] + 1;
         else if ( value = 'DELIVERY ADDRESS   ' ) then
      total_deladdr[ column_ord ] = total_deladdr[ column_ord ] + 1;
         else if ( value = 'CITY               ' ) then
      total_city[ column_ord ] = total_city[ column_ord ] + 1;
         else if ( value = 'POSTAL CODE        ' ) then
      total_postal[ column_ord ] = total_postal[ column_ord ] + 1;

         else if ( value = 'PHONE              ' ) then
      total_phone[ column_ord ] = total_phone[ column_ord ] + 1;

         else if ( value = 'UNKNOWN            ' ) then
      total_unkn[ column_ord ] = total_unkn[ column_ord ] + 1;

         else if ( value = 'NATIONAL ID        ' ) then
      total_natid[ column_ord ] = total_natid[ column_ord ] + 1;
         else if ( value = 'VEHICLE REGISTRATION' ) then
      total_vehreg[ column_ord ] = total_vehreg[ column_ord ] + 1;

    else if ( value = 'BLANK              ' ) then
      total_blnk[ column_ord ] = total_blnk[ column_ord ] + 1;
     column_ord = column_ord + 1;
  end; /* do for all columns */
  total_rows = total_rows + 1;
end;  /* For all sampled rows in the table */
```

```
    rc = 0;
/*********************************************************************/
/* SECTION F: Generate the output row describing this table's column  */
/*            information. There is one row generated for each        */
/*            table.column scanned.                                   */
/*********************************************************************/
    end_dt = datetime();
    start_fmt = PUT(start_dt, datetime18.);
    end_fmt   = PUT(end_dt, datetime18.);
    column_ord = 1;
    column_element       = column_list.front;
    data_type_element    = data_type_list.front;
    type_element         = type_list.front;
    column_size_element  = column_size_list.front;
    buffer_length_element= buffer_length_list.front;
    do while (column_ord <= stmt_identification.GETCOLUMNCOUNT());
      COLUMN_NAME   = column_element.c;
      DATA_TYPE     = data_type_element.d;
      TYPE_NAME     = type_element.c;
      COLUMN_SIZE   = column_size_element.d;
      BUFFER_LENGTH = buffer_length_element.d;

      if((total_nonchar[ column_ord ] > 0) or (total_rows = 0)) then do;
       email_pct       =            0;
       iban_pct        =            0;
       network_pct     =            0;
       pay_pct         =            0;
       url_pct         =            0;
       cntry_pct       =            0;
       cntry2_pct      =            0;
       cntry3_pct      =            0;
       date_pct        =            0;
       geo_pct         =            0;
       indiv_pct       =            0;
       org_pct         =            0;
       deladdr_pct     =            0;
       city_pct        =            0;
       postal_pct      =            0;
       phone_pct       =            0;
       unkn_pct        =            0;
```

```
 natid_pct       =            0;
 vehreg_pct      =            0;
 blnk_pct        =            0;

   if( total_nonchar[ column_ord ] > 0 ) then
      error_text  = 'Skipping non-character column';
   else
      error_text  = 'No rows in table';
end;
else do;
  error_text  = '';
  /* Determine percentages */
      email_pct       =round(total_email[column_ord]/total_rows*100, 0.01);
      iban_pct        =round(total_iban[column_ord]/total_rows*100, 0.01);
      network_pct     =round(total_network[column_ord]/total_rows*100, 0.01);
      pay_pct         =round(total_pay[column_ord]/total_rows*100, 0.01);
      url_pct         =round(total_url[column_ord]/total_rows*100, 0.01);
      cntry_pct       =round(total_cntry[column_ord]/total_rows*100, 0.01);
      cntry2_pct      =round(total_cntry2[column_ord]/total_rows*100, 0.01);
      cntry3_pct      =round(total_cntry3[column_ord]/total_rows*100, 0.01);
      date_pct        =round(total_date[column_ord]/total_rows*100, 0.01);
      geo_pct         =round(total_geo[column_ord]/total_rows*100, 0.01);
      indiv_pct       =round(total_indiv[column_ord]/total_rows*100, 0.01);
      org_pct         =round(total_org[column_ord]/total_rows*100, 0.01);
      deladdr_pct     =round(total_deladdr[column_ord]/total_rows*100, 0.01);
      city_pct        =round(total_city[column_ord]/total_rows*100, 0.01);
      postal_pct      =round(total_postal[column_ord]/total_rows*100, 0.01);
      phone_pct       =round(total_phone[column_ord]/total_rows*100, 0.01);
      unkn_pct        =round(total_unkn[column_ord]/total_rows*100, 0.01);
      natid_pct       =round(total_natid[column_ord]/total_rows*100, 0.01);
      vehreg_pct      =round(total_vehreg[column_ord]/total_rows*100, 0.01);
      blnk_pct        =round(total_blnk[column_ord]/total_rows*100, 0.01);
end;
/* Write out the record */
output;
/* Reset all stats */
            total_email[ column_ord ]         = 0;
            total_iban[ column_ord ]          = 0;
            total_network[ column_ord ]      = 0;
            total_pay[ column_ord ]           = 0;
```

```
                    total_url[ column_ord ]          = 0;
                    total_cntry[ column_ord ]        = 0;
                    total_cntry2[ column_ord ]       = 0;
                    total_cntry3[ column_ord ]       = 0;
                    total_date[ column_ord ]         = 0;
                    total_geo[ column_ord ]          = 0;
                    total_indiv[ column_ord ]        = 0;
                    total_org[ column_ord ]          = 0;
                    total_deladdr[ column_ord ]      = 0;
                    total_city[ column_ord ]         = 0;
                    total_postal[ column_ord ]       = 0;
                    total_phone[ column_ord ]        = 0;
                    total_unkn[ column_ord ]         = 0;
                    total_natid[ column_ord ]        = 0;
                    total_vehreg[ column_ord ]       = 0;
                    total_blnk[ column_ord ]         = 0;
                    total_nonchar[ column_ord ]      = 0;

      column_element        = column_element.get_next();
      data_type_element     = data_type_element.get_next();
      type_element          = type_element.get_next();
      column_size_element   = column_size_element.get_next();
      buffer_length_element = buffer_length_element.get_next();

      column_ord            = column_ord + 1;

    end; /* cutting output records */

/*********************************************************************/
/* SECTION G: Clean up our resources and process the next table in    */
/*            the dictionary.                                        */
/*********************************************************************/
cleanup:
    stmt_identification.Delete();

next_table:
    if( rc ne 0 ) then do;
      /* output the error record with time stats */
      end_dt = datetime();
```

```
        /* Note:  start and end date should be the same for all column  */
        /*        information rows for each table                      */
        start_fmt = PUT(start_dt, datetime18.);
        end_fmt   = PUT(end_dt, datetime18.);
        output;
      end;

      column_list.delete();
      data_type_list.delete();
      type_list.delete();
      column_size_list.delete();
      buffer_length_list.delete();

      stmt_columns.Delete();

  next_table_loop:
    end;  /* end 'run' */
  endthread;


  /*****************************************************************************/
  /* EDIT HERE EDIT HERE EDIT HERE EDIT HERE EDIT HERE EDIT HERE EDIT HERE        */
  /* SECTION H: This is the primary DS2 program that runs the thread              */
  /* program above and saves the results in idanalysis. Save results in          */
  /* a catalog and schema of your choice in the notation catalog.schema.tablename */
  /* example: orion.ds2_output.idanalysis                                        */
  /* If you are testing in Federation Server SQL Console, remember to add the     */
  /* DSN for the results.                                                        */
  /* You can modify the number of threads below. The default is 4                */
  /*****************************************************************************/

 /* Example catalog.schema.tablename for results        */
 /*    data orion.ds2_output.idanalysis(overwrite=yes); */
     data idanalysis(overwrite=yes);
     dcl thread tpgm tdata;
      method run();
      set from tdata threads=4 ;
      output;
      end;
enddata;
```

ACKNOWLEDGMENTS

A tremendous thanks to Brian Hess (Sr. Manager, Software Development) and Tom Cole (Principal Software Developer), Platform Research and Development, SAS Institute Inc. for their contribution to the section, *Scenario 3: Identifying personal data with DS2 in SAS Federation Server Manager*. They very kindly compiled the DS2 program and they have provided fantastic support on SAS Federation Server.

Many thanks also to Jens Dahl Mikkelsen (Principal Advisor, Nordic Technology Practice) for his initial review of this paper.

RECOMMENDED READING

- *SAS Federation Server*

  http://www.sas.com/en_us/software/data-management/data-federation.html

  http://support.sas.com/documentation/onlinedoc/fedserver/index.html

- *SAS Data Management Studio*

  http://www.sas.com/en_us/software/data-management/data-federation.html

  http://support.sas.com/documentation/onlinedoc/dfdmstudio/index.html

- *SAS and EU GDPR*

  http://www.sas.com/da_dk/insights/articles/risk-fraud/eu-data-regulation.html

  http://www.sas.com/en_gb/offers/17q1/general-data-protection-regulation.html

CONTACT INFORMATION

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

Cecily Hoffritz
Principal Advisor
SAS Institute Denmark
cecily.hoffritz@sas.com
http://www.sas.com

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

Other brand and product names are trademarks of their respective companies.