Paper SAS3370-2019

Unlocking Your Data With SAS/ACCESS® Interface to Salesforce

Kevin Kantesaria, Mason Morris, SAS Institute Inc., Cary, NC

ABSTRACT

Are you a SAS® programmer searching for insights within Salesforce data? Accessing and analyzing your information is now easier than ever before with the all-new SAS/ACCESS® Interface to Salesforce. This paper provides you with an overview of how to explore Salesforce objects as if they were native SAS data sets. Learn how to directly execute the SOQL language used by Salesforce by using the SQL procedure, and take advantage of the advanced features of implicit SQL to let SAS do some of the work for you. With the help of SAS/ACCESS Interface to Salesforce, you'll begin to understand your data in no time!

INTRODUCTION

SAS/ ACCESS to Salesforce enables you to connect to a Salesforce organization with minimal configuration steps and no additional software. This paper will guide you through the configuration steps to connect to Salesforce and will then demonstrate how to navigate your data and work with it natively from SAS.

This paper will use screenshots in the Salesforce Lightning web interface, though the same steps can be accomplished using Salesforce Classic.

ACCESS TO SALESFORCE: GETTING STARTED

CONNECTING

Before you can begin, your Salesforce administrator will need to enable API permissions for your Salesforce user's profile. This allows SAS to act as a client that will communicate directly with Salesforce. For optimum security, SAS recommends that your administrator create an isolated profile and associated user account for connecting. This will allow for more granular control over the account's security.

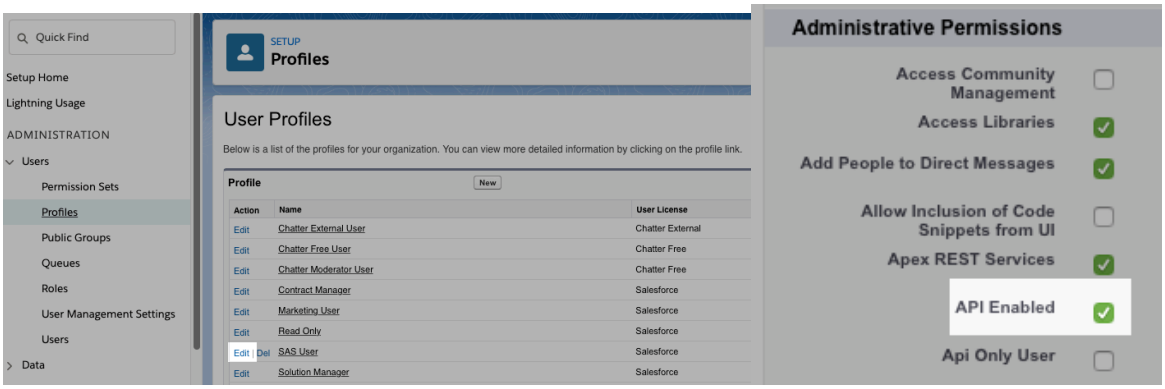


Figure 1: Administrative Permisssions for a User Profile

If your administrator has enabled two-factor authentication, you will also need a "security token" to connect. When making a LIBNAME connection in SAS, you will need to append this string of characters directly to the end of your password. You cannot view your security token from the dashboard. Therefore, if you do not already have it at hand, you will need to

reset it. A new security token will then be emailed to you. You can do this by first going to your user settings and then locating the "Reset My Security Token" link.

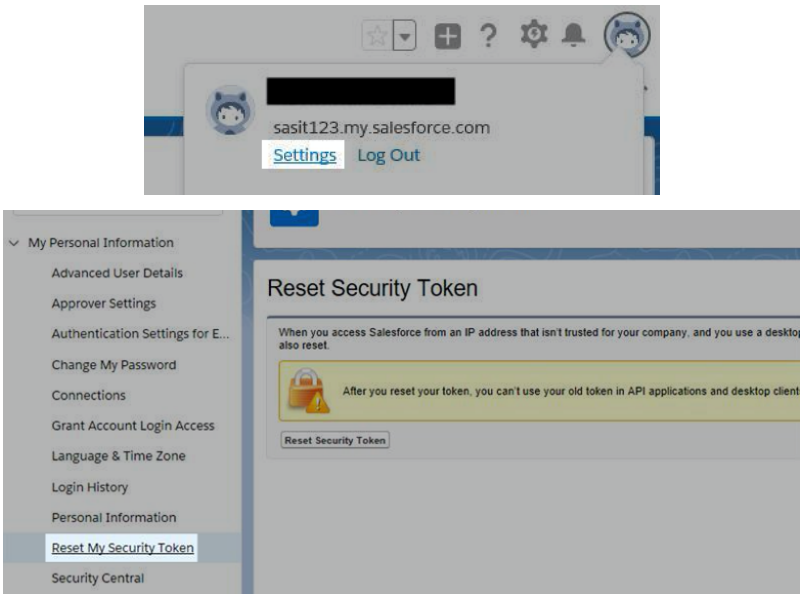


Figure 2: Reset Security Tokens

Once you have received your security token, append it directly to the end of your password (no space between the two) when creating a LIBNAME connection.

SECURITY FEATURES

When you make a connection using SAS/ACCESS to Salesforce without any non-required LIBNAME options, the Salesforce engine will use a secure TLS connection with server-side authentication. This means that your session data will be encrypted over the wire, and SAS will verify the identify of Salesforce using a public certificate signed by a code authority.

If you want a more secure connection, SAS/ ACCESS to Salesforce also supports mutual authentication. Once enabled, this feature allows you to upload your own certificate signed by a code authority to Salesforce. In addition to SAS verifying Salesforce's public certificate, Salesforce will then verify your identity as well on each connection and subsequent query.

There are several steps you will need to take to correctly configure mutual authentication. To begin, you must generate a Certificate Signing Request (CSR) for the server on which SAS will run. Once an external signer on Salesforce's list of trusted root certificate authorities has signed your certificate, you will need to import the certificate and key into environment. Let's walk through these steps.

Generating a Certificate for Mutual Authentication

To generate a certificate for mutual authentication:

1. Create a Certificate Signing Request (CSR) for the server that needs to be secured. Digicert provides a wizard that you can use for generating an OpenSSL command, which will then need to be run on the server to be secured (https://www.digicert.com/easy-csr/openssl.htm). You should now have a .csr file and a private .key file. **It is very important to keep the private key in a secured location. If you lose the key, you must restart at step 1.**

```
openssl req -new -newkey rsa:2048 -nodes -out <request.csr> -keyout
<private.key> -subj "/C=US/ST=North Carolina/L=Cary/O=Example Institute
Inc./CN=example.example.com"
```

2. Present the CSR to an external signer on Salesforce's list of trusted root certificate authorities (https://cs32.salesforce.com/cacerts.jsp). You should get back a signed certificate, which is in .p7b or .pem format. If the certificate is in .p7b format, it should be converted to a .pem file, which you can do with this command:

```
openssl pkcs7 -print_certs -in <certificate.p7b> -out <certificate.pem>
```

3. Make a copy of the .pem file and change the extension from ".pem" to ".crt". This is the certificate file you will upload to Salesforce.
4. Create a keystore in PKCS format using the certificate and private key. You can do so by executing this command:

```
openssl pkcs12 -export -out <access_keystore.p12> -inkey <private.key>
-in <certificate.crt>
```

You will now have three important files:

- private.key - the SAS server's private key
- certificate.crt - the certificate in a format Salesforce recognizes
- access_keystore.p12 - an encrypted file containing the private key and certificate

Configuring the Salesforce Instance for Mutual Authentication via the Web Interface

To configure the Salesforce instance for mutual authentication via the Web interface:

1. As an administrator, enter the "Setup" area of the Salesforce website. Under the sidebar heading "Security," locate "Certificate and Key Management." find the sub-section titled "Mutual Authentication Certificates." NOTE: If you do not see this sub-section, contact your Salesforce administrator. This functionality must be enabled by Salesforce, usually through a request in their support system

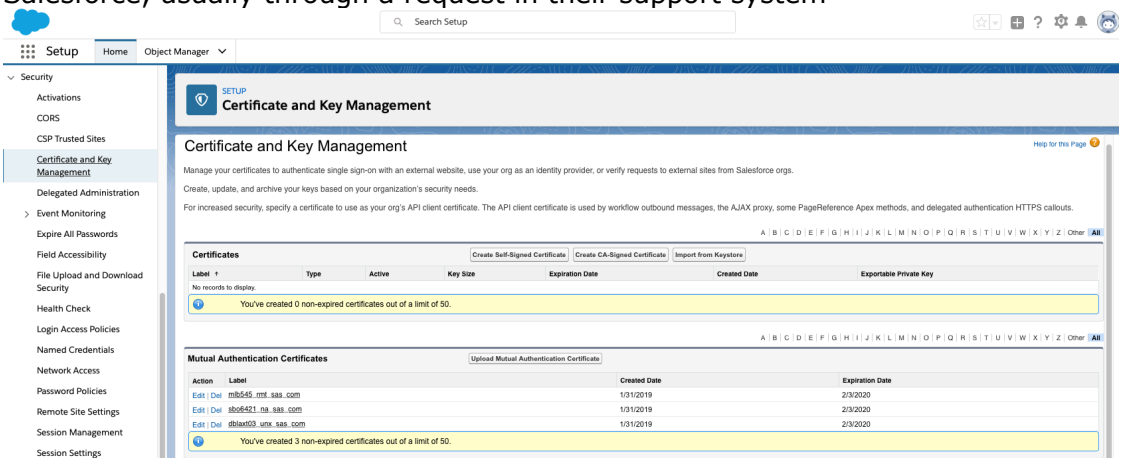


Figure 3: Certificate and Key Management

2. Upload the signed .crt file to the "Mutual Authentication Certificates" sub-table. Give it a label and unique name (if these fields are not auto-populated). Click "Save."

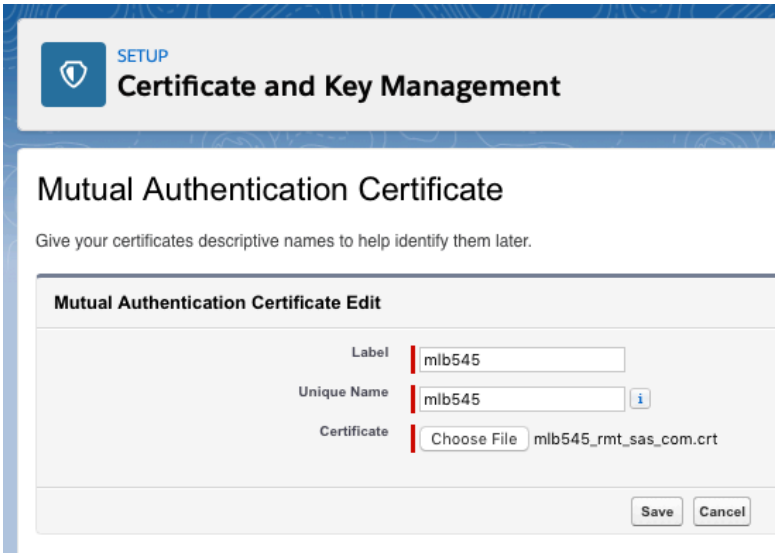


Figure 4: Mutual Authentication Certificate

3. You will need a separate profile in Salesforce for the user account that will access Salesforce with mutual authentication. That profile should have the permissions "API Only" and "Enforce SSL/ TLS Mutual Authentication" enabled.

Configuring SAS (UNIX) for Mutual Authentication

To configure SAS (UNIX) for mutual authentication:

1. Copy the .p12 keystore file to the server running SAS. Keep it in a secure location with Read permissions open for the SAS user.
2. When making a LIBNAME connection to SAS, use the option MUTUAL_AUTH=yes. Provide the location of your keystore containing a certificate and the key. You can also provide a keystore password if you created yours with one.

```
libname x sasioslf USER="" PASS="myPass" MUTUAL_AUTH=yes
CERT_PATH="<keystore.p12>" CERT_PASS="oPeNsEsAmE";
```

Configuring SAS (Windows) for Mutual Authentication

To configure SAS (Windows) for mutual authentication:

1. Copy the .p12 keystore file to the server running SAS.
2. Follow SAS documentation (http://support.sas.com/documentation/cdl/en/secref/69831/HTML/default/viewer.htm#n12036intelplatform00install.htm) to add the keystore to your Windows Certificate Manager. When presented with the Certificate Import Wizard, be sure to select "Current User" as the "Store Location," and select the "Personal" certificate store.

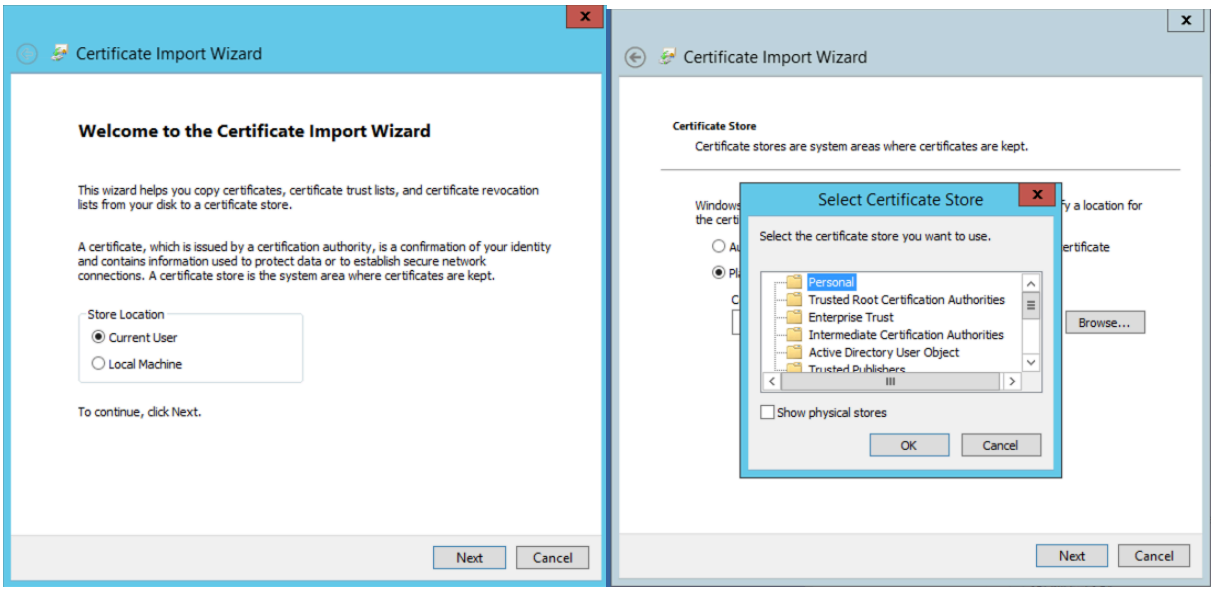


Figure 5: Certificate Import Wizard

MONITORING API USAGE

Salesforce limits your organization's API usage. SAS is limited in the number of times that it can make requests to Salesforce, including requests for data fetches. SAS has optimized some queries to work around this, but it is important to understand these limitations to make the best use of your API allocation.

If you want to receive detailed notes in your SAS log explaining API usage against your organization during execution of SAS statements, you can use the API_TRACE LIBNAME option. Enabling this option might cause a small number of extra API calls to be made, and it will result in verbose logging that might slow down SAS processing.

```
libname x sasioslf user="" pass="" API_TRACE=yes;

proc sql;
  SELECT Name, Description, Industry FROM x.Account;
quit;
```

```
NOTE: Writing HTML Body file: sashtml.htm
NOTE: API Trace: Received statement off connection. Usage: 51.4%,
41103 calls used out of 80000 allocated.
NOTE: API Trace: Table existence check. Usage: 51.4%, 41104 calls
used out of 80000 allocated.
NOTE: PROCEDURE SQL used (Total process time):
      real time           8.00 seconds
      cpu time            0.78 seconds
```

Salesforce only provides SAS with information about the overall organization usage, so this reporting includes usage for any processes connecting to Salesforce at the time the SAS job is running. If another process is also connected to the same Salesforce organization, the API calls it consumes will also count toward the reported usage. If you would like to work

around this, consider using a Salesforce sandbox instance while isolating the API usage solely to SAS.

SQL Processing in SAS

When you execute ANSI SQL using PROC SQL or PROC FEDSQL, the query is parsed and mapped to the corresponding Salesforce SOQL(Salesforce Object Query Language). If an equivalent match in the Salesforce SOQL language can be found, SAS will take advantage of it by transforming your query to implicitly pass it down to Salesforce. Here's a simple example:

```
proc sql;
  SELECT COUNT(*) FROM x.Account;
quit;
```

SAS recognizes that the COUNT() syntax is supported by SOQL, but that the wildcard symbol is not. SAS will transform your query into an equivalent SOQL statement, "SELECT COUNT(Id) FROM Account," and pass it to the database, which returns the result.

The following aggregate functions will be passed implicitly to Salesforce, as they are supported natively by the SOQL syntax:

```
AVG(fieldName)
COUNT(), COUNT(*), and COUNT(fieldName)
COUNT(DISTINCT fieldName)
MIN(fieldName)
MAX(fieldName)
SUM(fieldName)
```

In addition, the following clauses will be implicitly passed to Salesforce:

```
LIMIT
GROUP BY
WHERE
ORDER BY
OFFSET
```

Explicit Pass-Through

With explicit pass-through, you can pass SOQL statements directly to Salesforce and materialize the results natively in SAS. For some queries, this can be the most conservative method of execution with respect to API consumption. Queries submitted in this way must match the SOQL syntax guidelines (https://developer.salesforce.com/docs/atlas.en-us.soql_sosl.meta/soql_sosl/sforce_api_calls_soql_select.htm).

There are several ways to execute SQL directly using SAS/ACCESS to Salesforce. If you've defined a LIBNAME, for example x, you can use this syntax:

```
proc sql;
connect using x;
select * from connection to x ( <SOQL with data returned> );
disconnect from x;
quit;

proc fedsql iptrace;
select * from connection to x ( <SOQL with data returned> );
```

```
execute ( <SOQL without data returned> );
quit;
```

If you prefer to use a connection string, you can use this syntax:

```
proc fedsql iptrace nolibs
prompt="DRIVER=SQL;CONOPTS=(DRIVER=SFORCE;UID=<uid>;PWD=<pwd>)";
quit;
```

MAPPING SALESFORCE BEHAVIORS TO SAS

HANDLING CUSTOM OBJECTS AND FIELDS

Salesforce includes several pre-defined "standard" objects with pre-defined fields. It also enables users to create custom fields and custom objects. These user-defined objects and fields are suffixed with "__c". Traditionally, the SOQL syntax requires users to use this suffix when referencing any custom objects or tables.

As an added convenience, SAS enables users to reference tables and objects without a suffix. This behavior can be controlled using the "NATIVE_NAMES" LIBNAME option, which, when set to "yes", emulates the SOQL syntax and requires the "__c" suffix. By default, it is set to "no", which means you do not need to use the suffix.

NOTE: If you are using explicit SQL to pass SOQL directly to Salesforce, you will need to fully qualify any custom objects or fields by including the suffix "__c".

DATA TYPES

SAS handles each of the Salesforce data types by converting them to the closest equivalent in SAS. If you are using PROC SQL or PROC FEDSQL, your data will be read in using one of the following conversions:

Salesforce	PROC SQL	PROC FEDSQL
date	DATE	DATE
byte	DOUBLE	TINYINT
address	CHAR	VARCHAR
base64	CHAR	VARCHAR
encryptedstring	CHAR	VARCHAR
location	CHAR	VARCHAR
currency	DOUBLE	DOUBLE
dateTime	DATETIME	TIMESTAMP
email	CHAR	VARCHAR
masterrecord	CHAR	VARCHAR
reference	CHAR	VARCHAR
multipicklist	CHAR	VARCHAR
picklist	CHAR	VARCHAR

percent	DOUBLE	DOUBLE
phone	CHAR	VARCHAR
combobox	CHAR	VARCHAR
anyType	CHAR	VARCHAR
ID	CHAR	VARCHAR
DataCategoryGroupReference	CHAR	VARCHAR
double	DOUBLE	DOUBLE
int	DOUBLE	INTEGER
JunctionIdList	CHAR	VARCHAR
textarea	CHAR	VARCHAR
string	CHAR	VARCHAR
time	TIME	TIME
Boolean	DOUBLE	TINYINT
url	CHAR	VARCHAR

Table 1: Salesforce Data Type Conversion

SALESFORCE METADATA

Salesforce objects automatically include some fields that contain metadata you might consider extraneous, especially when creating reports or graphs. SAS has identified many of these fields, including system fields, and hides them from the default view. If you want to disable this filtering, you can use the SHOW_METADATA LIBNAME option. This can be especially useful if you are using SAS in conjunction with the Salesforce Apex language, or if you are interested in metadata regarding table access or creation and modification information. The following columns are considered metadata in this first release of SAS/ ACCESS to Salesforce:

```
CreatedBy
CreatedById
CreatedDate
CurrencyIsoCode
Id
IsDeleted
LastActivityDate
LastModifiedById
LastModifiedDate
LastReferencedDate
LastViewedDate
MasterRecordId
OwnerId
RecordTypeId
SystemModstamp
```

SOFT DELETES

In Salesforce, when a record is deleted, it is not immediately removed from the object. Instead, a field named "IsDeleted" is set to 1, and the data is hidden. After 15 days, it will be removed.

If you want to view soft-deleted data, SAS has a LIBNAME option named “SHOW_DELETED” to show these records:

```
libname x sasioslf user="" pass="" show_deleted=yes;
```

After you run the LIBNAME statement, you will still see columns that are supposed to be deleted. For example, in the Accounts object, there is a deleted record with the value "SAS Institute Inc." However, When you run PROC PRINT, you will still see the deleted record. You will also have access to the metadata column IsDeleted:

```
proc print data=x.Account;
run;
```

Obs	Id	IsDeleted	MasterRecordId	Name	Type	ParentId	BillingStreet	BillingCity	BillingState	BillingPostalCode	BillingCountry	Bil
1	0015A00002AkmDoQAJ	1		SAS Institute Inc	Partner		100 SAS Campus Dr	Cary	North Carolina	27513-2414	United States	
2	0015A000025kcnYQAQ	0		Acme (Sample)	Prospect		10 Main Rd.	New York	NY	31349	USA	
3	0015A000025kcnZQAQ	0		Global Media (Sample)	Prospect		150 Chestnut Street	Toronto	Ontario	L4B 1Y3	Canada	

Figure 6: Object Shows Deleted Record With IsDeleted Field Set to 1

Although the first observation was deleted, it is displayed with this option. This can be useful to quickly recover your data using SAS instead of navigating the Salesforce recycle bin.

WILDCARD SELECTS

Salesforce does not support wildcard SELECT statements with its SOQL syntax, but SAS emulates the behavior in both native SQL and PROC SQL. When SAS identifies a query with a wildcard SELECT, it will identify the relevant Salesforce object and expand the query to include the full list of fields in it, depending on usage of the SHOW_METADATA option.

This can result in additional API usage for your organization. If you are trying to minimize this, be sure to narrow your queries down by specifying exact column names.

PUTTING EVERYTHING TOGETHER

Once you have a Salesforce account, you can now connect to Salesforce using SAS. For example, you can create a LIBNAME connection using the following:

```
libname x sasioslf user=”<username>” pass=”<password>”;
```

Remember that when you use your password, you will need to append the API token to the end of your password.

Now you can use a sample table, Account, to quickly create a chart in SAS.

You can quickly see the benefits of using SAS/ACCESS to Salesforce in being able to use a wildcard statement in SELECT. After submitting the following statements, the data shown in Figure 8 is displayed:

```
proc sql;
  select * from x.Account;
quit;
```

Account Name	Account Type	Parent Account ID	Billing Street	Billing City	Billing State/Province	Billing Zip/Postal Code	Billing Country	Billing Latitude	Billing Longitude	Billing Geocode Accuracy	Shipping Street	Shipping City	Shippi State/P
Acme (Sample)	Prospect		10 Main Rd.	New York	NY	31349	USA	.	.		10 Main Rd.	New York	NY
Global Media (Sample)	Prospect		150 Chestnut Street	Toronto	Ontario	L4B 1Y3	Canada	.	.		150 Chestnut Street	Toronto	Ontario

Figure 8: PROC SQL Results Using SELECT *

There are many more columns in this table, but they are irrelevant for now.

Now we can do whatever SAS supports with the data. Here a simple pie chart is created showing employee numbers for the two accounts.

```
proc gchart data=x.Account;
  pie Name / sumvar=NumberOfEmployees Coutline=black;
run;quit;
```

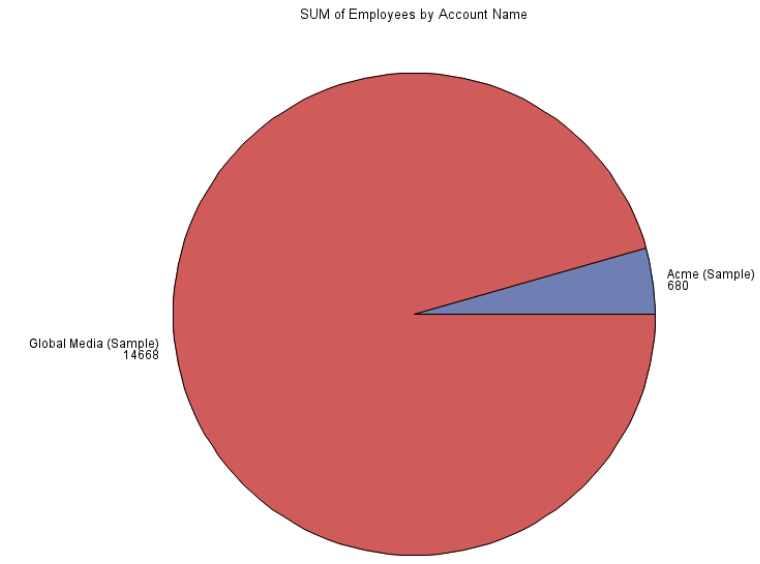


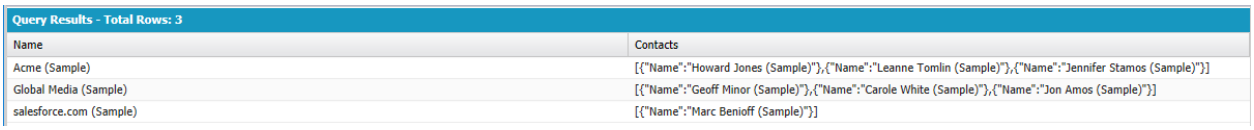
Figure 9: Pie Chart Showing Employee Numbers

As of now, SOQL is supported with SAS. To submit direct SOQL statements, you can use FedSQL.

Below is an example of an inner join using FedSQL to explicitly pass-through some SOQL to Salesforce.

```
proc fedsql;
  select * from connection to x (SELECT Account.Name, (SELECT Contact.Name
  FROM contacts) FROM Account);
quit;
```

As you can see, the SOQL statement is included in the parenthesis after the "x". Here are what the results look like in the Salesforce Developer Console:

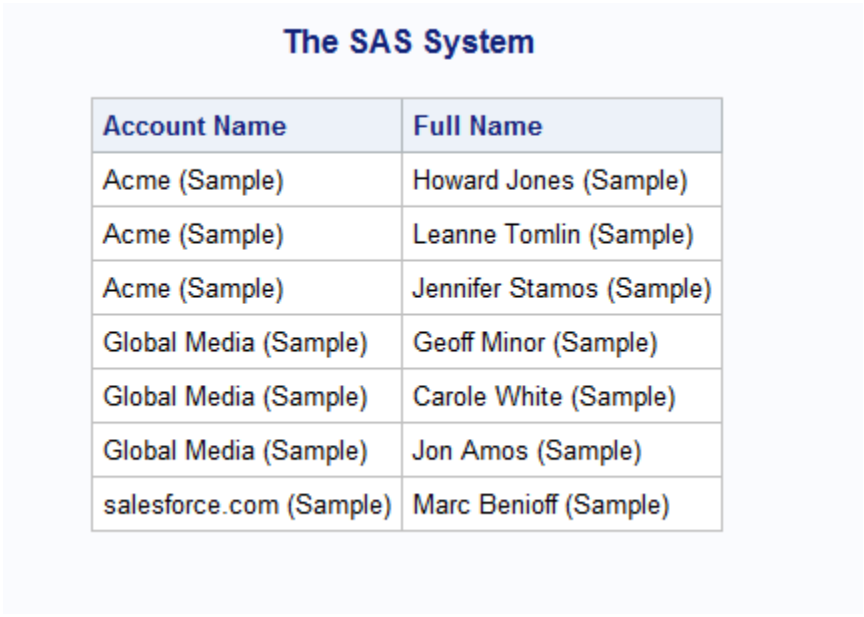


Query Results - Total Rows: 3

Name	Contacts
Acme (Sample)	[{"Name":"Howard Jones (Sample)"},{"Name":"Leanne Tomlin (Sample)"},{"Name":"Jennifer Stamos (Sample)"}]
Global Media (Sample)	[{"Name":"Geoff Minor (Sample)"},{"Name":"Carole White (Sample)"},{"Name":"Jon Amos (Sample)"}]
salesforce.com (Sample)	[{"Name":"Marc Benioff (Sample)"}]

Figure 10: Results Shown in Salesforce Developer Console

And here is what it looks like in SAS:



The SAS System

Account Name	Full Name
Acme (Sample)	Howard Jones (Sample)
Acme (Sample)	Leanne Tomlin (Sample)
Acme (Sample)	Jennifer Stamos (Sample)
Global Media (Sample)	Geoff Minor (Sample)
Global Media (Sample)	Carole White (Sample)
Global Media (Sample)	Jon Amos (Sample)
salesforce.com (Sample)	Marc Benioff (Sample)

Figure 11: Results Shown in SAS

Once the data is in SAS, one option is to treat the data relationally if you don't want to create relationships in Salesforce for your objects. For example, if the Account and Contact tables both had the same Names, you could use the following code to create an inner join:

```
proc sql;
  select * from x.Account, x.Contact
    where Account.Name=Contact.Name;
quit;
```

CONCLUSION

The SAS/ACCESS Interface to Salesforce is a versatile product that lets you interact with Salesforce using the traditional SAS methods and mechanisms. Now you can manipulate and analyze your data using the traditional SOQL methods and the full suite of procedures and applications that SAS offers. This paper highlights the biggest quirks and advantages that apply to SAS/ACCESS Interface to Salesforce that will allow you to comfortably connect with your data. For a bigger picture and more details, be sure to read the SAS/ACCESS Interface to Salesforce documentation for a more comprehensive guide.

REFERENCES

Salesforce.com Inc. April 2018. SOQL SELECT Syntax.Available https://developer.salesforce.com/docs/atlas.en-us.soql_sosl.meta/soql_sosl/sforce_api_calls_soql_select.htm.

Patterson, Pat. "Salesforce Mutual Authentication – Part 1: The Basics". Available https://blog.superpat.com/2018/01/25/salesforce-mutual-authentication-part-1-the-basics/. Last modified January 25, 2018. Accessed February 28, 2019.

Pidikiti Praneel. " Mutual Authentification Salesforce". Available https://www.aboveandbeyondcloud.com/mutual-authentification-salesforce/. Last Modified June 10, 2015. Accessed February 28, 2019.

ACKNOWLEDGMENTS

The authors extend their thanks and gratitude to everyone who contributed in the development, support, documentation, and enablement of SAS/ ACCESS Interface to Salesforce. In addition, they would like to extend thanks to the following individuals who provided their time and expertise for this paper:

Aparna Amin, SAS Institute Inc.

Salman Maher, SAS Institute Inc.

Joel Odom, SAS Institute Inc.

Bill Oliver, SAS Institute Inc.

Julia Schelly, SAS Institute Inc.

Joe Schluter, SAS Institute Inc.

Julian Taylor, SAS Institute Inc.

Brad Thompson, SAS Institute Inc.

CONTACT INFORMATION

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

Mason Morris
500 SAS Campus Drive
Cary, NC 27513
SAS Institute Inc.
Mason.Morris@sas.com
http://www.sas.com

Kevin Kantesaria
500 SAS Campus Drive
Cary, NC 27513
SAS Institute Inc.
Kevin.Kantesaria@sas.com
http://www.sas.com