Paper SAS4468-2020

SAS® and Amazon Redshift: Overview of Current Capabilities

Nicolas Robert, SAS Institute Inc.

ABSTRACT

Amazon Redshift has become a very popular database management system in modern cloud infrastructures. It provides flexibility and agility, and it can scale easily and save costs. More and more companies are using Redshift as their new data warehousing tool. How does SAS® fit in this context? How do SAS users work efficiently with Redshift data? This paper gives an overview of the current access and integration capabilities between SAS products and Amazon Redshift. We cover this from both SAS® Viya® and SAS®9 standpoints, including news from the latest releases.

INTRODUCTION

SAS provides a rich set of capabilities when Amazon Redshift comes into play. Whether you are still on SAS 9 or have already moved to SAS Viya, you want to load data quickly into Redshift, process Redshift data efficiently without involving data movement, extract data from Redshift as fast as possible, and load it in SAS or in SAS® Cloud Analytic Services (CAS), the SAS Viya in-memory analytics engine, so that you can analyze your cloud data with the most advanced analytics platform.

This paper describes these different techniques used in SAS to work with Redshift data:

- SAS 9 tasks
  - Loading data in Redshift
  - Processing Redshift data
  - Extracting Redshift data
- SAS Viya tasks
  - Loading Redshift data in CAS
  - Processing Redshift data from CAS
  - Saving CAS data to Redshift

WORK WITH REDSHIFT DATA FROM SAS 9

LOADING DATA IN REDSHIFT

The first thing that you may have to do (or the last thing, depending on your project!), is to load data from SAS to Redshift. There are various options.

Standard load

Without specifying any options, you can load data from SAS to Redshift. Will it be efficient? Well, it does the job with small amounts of data, but if you are dealing with huge amounts of data, then you need to find some alternatives. Basically, Redshift inserts are slow.

Here is a sample log of a standard load:

```
82   /* Standard Redshift library */
83   libname myrs redshift server="myredshift.amazonaws.com"
84                         database="mydb" schema="public"
85                         user="myuser" password=XXXXXXXXXX ;
NOTE: Libref MYRS was successfully assigned as follows:
      Engine:        REDSHIFT
      Physical Name: myredshift.amazonaws.com
86
87   proc append base=myrs.lineorder
88               data=local.lineorder(obs=50000) ;
89   run ;
NOTE: Appending LOCAL.LINEORDER to MYRS.LINEORDER.
NOTE: There were 50000 observations read from the data set LOCAL.LINEORDER.
NOTE: 50000 observations added.
NOTE: The data set MYRS.LINEORDER has . observations and 17 variables.
NOTE: PROCEDURE APPEND used (Total process time):
      real time           54.37 seconds
      cpu time            1.82 seconds
```

It takes 54 seconds to load 50,000 records, so it will take 4.5 hours to load a table of 15 million rows.

Using buffer options

For years, SAS/ACCESS has been providing these options to control various buffers: READBUFF, INSERTBUFF, UPDATEBUFF.

The default value for INSERTBUFF with Redshift is 250. Increasing it to a much larger value can give spectacular results.

Here is how to set a different INSERTBUFF value while loading data in Redshift:

```
135   proc append base=myrs.lineorder(insertbuff=4096)
136               data=local.lineorder(obs=50000) ;
137   run ;
NOTE: Appending LOCAL.LINEORDER to MYRS.LINEORDER.
NOTE: There were 50000 observations read from the data set LOCAL.LINEORDER.
NOTE: 50000 observations added.
NOTE: The data set MYRS.LINEORDER has . observations and 17 variables.
NOTE: PROCEDURE APPEND used (Total process time):
      real time           3.80 seconds
      cpu time            0.43 seconds
```

With an appropriate value for INSERTBUFF, we managed to load the same 50,000 records in less than 4 seconds, and this entire table (15 million rows) in about 16 minutes.

Bulk Loading

SAS/ACCESS® Interface to Amazon Redshift also provides bulkload capabilities, starting with SAS 9.4M4. Unlike other bulkload capabilities available with other databases, which sometimes require additional software components such as SQL*Loader for Oracle, the Redshift bulk-load capability relies only on the use of AWS S3 as a staging area for moving data.

When the bulk-load facility is active, SAS exports the SAS data set as a set of text files (with a .dat extension) using a default delimiter (the bell character), loads them in AWS S3

using the AWS S3 API, and runs a Redshift COPY command to load the text files into an existing Redshift table. This method is particularly efficient.

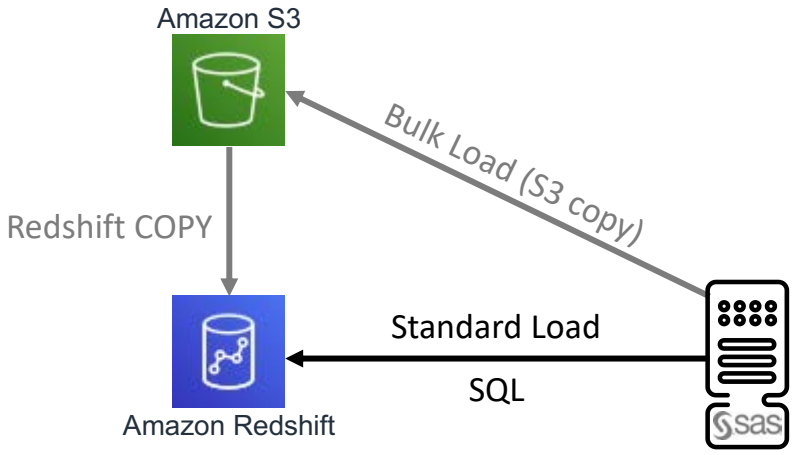

Figure 1 - SAS to Amazon Redshift Bulk Loading Process

From a code perspective, it looks like this:

```
84    proc append base=myrs.lineorder(bulkload=yes
84  ! bl_bucket="sas-bucket/redshift/temp_bulk_loading")
85                data=local.lineorder(obs=5000000) ;
86    run ;
NOTE: Appending LOCAL.LINEORDER to MYRS.LINEORDER.
NOTE: There were 5000000 observations read from the data set
LOCAL.LINEORDER.
NOTE: 5000000 observations added.
NOTE: The data set MYRS.LINEORDER has . observations and 17 variables.
  62 1571739581 no_name 0 APPEND
REDSHIFT_15: Executed: on connection 2 63 1571739581 no_name 0 APPEND
copy "public".LINEORDER
("lo_orderkey","lo_linenumber","lo_custkey","lo_partkey","lo_suppkey","lo_o
rderdate","lo_orderpriority","lo_shippriority","lo_quantity","lo_extendedpr
ice","lo_ordertotalprice","lo_discount","lo_revenue","lo_supplycost","lo_ta
x","lo_commitdate","lo_shipmode") FROM 's3://sas-
bucket/redshift/temp_bulk_loading/SASRSBL_1AC33569-9638-3B47-9714-
E1F5D307B619.manifest' ACCESS_KEY_ID '' SECRET_ACCESS_KEY '' SESSION_TOKEN
'' DELIMITER '\007' MANIFEST  REGION 'us-east-1'  64 1571739581 no_name 0
APPEND
  65 1571739581 no_name 0 APPEND
NOTE: PROCEDURE APPEND used (Total process time):
      real time           31.91 seconds
      cpu time            20.52 seconds
```

Bulk-load of 50,000 rows takes about a second, 5 million records takes about 32 seconds, and the whole table takes less than 90 seconds!

In order to use the Redshift bulk-load feature, you must properly set up the AWS keys. In the previous example, the AWS keys were defined in the user profile under an .aws sub-directory. SAS gets the AWS keys automatically, so you only need to use the BL_BUCKET option to specify the target AWS S3 bucket to store the temporary files. You could use additional bulk-load options to set your AWS profile, configuration file, or AWS keys directly in the program, although this is not recommended. For more information, refer to “Bulk Loading for Amazon Redshift” in *SAS/ACCESS® 9.4 for Relational Databases: Reference*.

These are some other options of interest:

- BL_NUM_DATA_FILES specifies the number of data files to create in order to contain the source data. This option is useful when your Redshift cluster is a multi-node cluster, because it allows multiple files to be loaded in parallel on multiple Redshift nodes
- BL_COMPRESS specifies whether to compress data files using the gzip format. This option is useful when your network bandwidth between SAS and Redshift is limited and your data files are very big, because it gzips the data files on the SAS machine before sending them to S3. This process is CPU-intensive on the SAS server.

PROCESS REDSHIFT DATA

When you have data residing in Redshift, you probably want to analyze it or manipulate it in different ways in order to build customized data sets that are required by your analytics processes. You can certainly move the data from Redshift down to SAS (and we'll discuss how to extract Redshift data in SAS in the next section) and use the full stack of SAS capabilities. But you might want to know about the SAS® In-Database features that enable SAS users to transparently work with Redshift data without moving the data. This topic focuses on efficiently processing Redshift data in place, minimizing data movement between SAS and Redshift.

Implicit Pass-Through

Implicit pass-through is the process of translating SAS code into equivalent data source-specific SQL code so that it can be passed directly to the data source for processing. Implicit pass-through improves query response time and enhances security.

A lot of SAS users use the SQL language to work with SAS and third-party data, through the SAS SQL procedure. If you don't specify any options, SAS tries to push the processing to the database as much as it can.

For example, consider the following SQL code against Redshift data:

```
/* Standard Redshift library */
libname myrs redshift server="myredshift.amazonaws.com"
                      database="mydb" schema="public"
                      user="myuser" password="XXXXXX" ;

/* SAS Library */
libname local "~/data" ;

proc sql ;
   create table local.orders as select *
   from myrs.part p, myrs.supplier s, myrs.customer c, myrs.dwdate d,
myrs.lineorder lo
   where p.p_partkey=lo.lo_partkey and
         s.s_suppkey=lo.lo_suppkey and
         c.c_custkey=lo.lo_custkey and
         d.d_datekey=lo.lo_orderdate ;
quit ;
```

In this code, the user wants to join multiple tables from Redshift and store the result in a SAS table. But where does this join process occur? In SAS, after having downloaded all of the tables from Redshift to SAS, or in Redshift?

If you set the right option prior to running your code, you can get some useful information in the SAS log about how your SAS code is transformed into database code:

```
options sastrace=',,,d' sastraceloc=saslog nostsuffix msglevel=i ;
```

This code returns the following:

```
REDSHIFT_31: Prepared: on connection 0
 select p."p_partkey", p."p_name", p."p_mfgr", p."p_category",
p."p_brand1", p."p_color", p."p_type", p."p_size", p."p_container",
s."s_suppkey", s."s_name", s."s_address", s."s_city", s."s_nation",
s."s_region", s."s_phone", c."c_custkey", c."c_name",
c."c_address", c."c_city", c."c_nation", c."c_region", c."c_phone",
c."c_mktsegment", d."d_datekey", d."d_date", d."d_dayofweek",
d."d_month", d."d_year", d."d_yearmonthnum", d."d_yearmonth",
d."d_daynuminweek", d."d_daynuminmonth", d."d_daynuminyear",
d."d_monthnuminyear", d."d_weeknuminyear", d."d_sellingseason",
d."d_lastdayinweekfl", d."d_lastdayinmonthfl", d."d_holidayfl",
d."d_weekdayfl", lo."lo_orderkey", lo."lo_linenumber", lo."lo_custkey",
lo."lo_partkey", lo."lo_suppkey", lo."lo_orderdate",
lo."lo_orderpriority", lo."lo_shippriority", lo."lo_quantity",
lo."lo_extendedprice", lo."lo_ordertotalprice", lo."lo_discount",
lo."lo_revenue", lo."lo_supplycost", lo."lo_tax", lo."lo_commitdate",
lo."lo_shipmode" from "public".PART p, "public".SUPPLIER s,
"public".CUSTOMER c, "public".DWDATE d, "public".LINEORDER lo where
(p."p_partkey" = lo."lo_partkey") and (s."s_suppkey" =
lo."lo_suppkey") and (c."c_custkey" = lo."lo_custkey") and (d."d_datekey" =
lo."lo_orderdate")


REDSHIFT_32: Executed: on connection 0
Prepared statement REDSHIFT_31

ACCESS ENGINE:  SQL statement was passed to the DBMS for fetching data.
```

In this case, the join happens in Redshift and the resulting data is fetched on SAS because the user asked for it.

Now consider the same code, but instead of creating a SAS table, the user wants to create a Redshift table:

```
proc sql ;
   create table myrs.orders as select *
   from myrs.part p, myrs.supplier s, myrs.customer c, myrs.dwdate d,
myrs.lineorder lo
   where p.p_partkey=lo.lo_partkey and
         s.s_suppkey=lo.lo_suppkey and
         c.c_custkey=lo.lo_custkey and
         d.d_datekey=lo.lo_orderdate ;
quit ;
```

This code returns the following:

```
REDSHIFT_44: Executed: on connection 1
CREATE  TABLE
"public".orders(p_partkey,p_name,p_mfgr,p_category,p_brand1,p_color,p_type,
p_size,p_container,s_suppkey,s_name,s_address,s_city,s_na
```

```
tion,s_region,s_phone,c_custkey,c_name,c_address,c_city,c_nation,c_region,c
_phone,c_mktsegment,d_datekey,d_date,d_dayofweek,d_month,
d_year,d_yearmonthnum,d_yearmonth,d_daynuminweek,d_daynuminmonth,d_daynumin
year,d_monthnuminyear,d_weeknuminyear,d_sellingseason,d_l
astdayinweekfl,d_lastdayinmonthfl,d_holidayfl,d_weekdayfl,lo_orderkey,lo_li
nenumber,lo_custkey,lo_partkey,lo_suppkey,lo_orderdate,lo
_orderpriority,lo_shippriority,lo_quantity,lo_extendedprice,lo_ordertotalpr
ice,lo_discount,lo_revenue,lo_supplycost,lo_tax,lo_commit
date,lo_shipmode) as ( select p."p_partkey", p."p_name", p."p_mfgr",
p."p_category", p."p_brand1", p."p_color", p."p_type",
p."p_size", p."p_container", s."s_suppkey", s."s_name", s."s_address",
s."s_city", s."s_nation", s."s_region", s."s_phone",
c."c_custkey", c."c_name", c."c_address", c."c_city", c."c_nation",
c."c_region", c."c_phone", c."c_mktsegment", d."d_datekey",
d."d_date", d."d_dayofweek", d."d_month", d."d_year", d."d_yearmonthnum",
d."d_yearmonth", d."d_daynuminweek", d."d_daynuminmonth",
d."d_daynuminyear", d."d_monthnuminyear", d."d_weeknuminyear",
d."d_sellingseason", d."d_lastdayinweekfl", d."d_lastdayinmonthfl",
d."d_holidayfl", d."d_weekdayfl", lo."lo_orderkey", lo."lo_linenumber",
lo."lo_custkey", lo."lo_partkey", lo."lo_suppkey",
lo."lo_orderdate", lo."lo_orderpriority", lo."lo_shippriority",
lo."lo_quantity", lo."lo_extendedprice", lo."lo_ordertotalprice",
lo."lo_discount", lo."lo_revenue", lo."lo_supplycost", lo."lo_tax",
lo."lo_commitdate", lo."lo_shipmode" from "public".PART p,
"public".SUPPLIER s, "public".CUSTOMER c, "public".DWDATE d,
"public".LINEORDER lo where (p."p_partkey" = lo."lo_partkey") and
(s."s_suppkey" = lo."lo_suppkey") and (c."c_custkey" = lo."lo_custkey") and
(d."d_datekey" = lo."lo_orderdate") )
```

This time, the whole process occurs in Redshift, with no data movement between Redshift and SAS. Implicit pass-through is also possible on multi-table operations involving multiple librefs. However, in order for this to work, these LIBNAME properties must match exactly for all of the Redshift libraries: user ID, password, server, database, port, and SQL functions option value. See the “Passing Joins to Amazon Redshift” in *SAS/ACCESS for Relational Databases*.

In addition to SQL implicit pass-through, note that DATA step allows some processing parts, especially filtering conditions, to be implicitly sent to the database for execution. What is interesting is that SAS automatically translates SAS functions to the corresponding Redshift functions.

```
82   data promos ;
83      set myrs.orders(where=(index(p_type,"PROMO")>0)) ;
84   run ;

REDSHIFT_48: Prepared: on connection 0
SELECT  "p_type", "p_partkey", "p_name", "p_mfgr", "p_category",
"p_brand1", "p_color", "p_size", "p_container", "s_suppkey",
"s_name", "s_address", "s_city", "s_nation", "s_region", "s_phone",
"c_custkey", "c_name", "c_address", "c_city", "c_nation",
"c_region", "c_phone", "c_mktsegment", "d_datekey", "d_date",
"d_dayofweek", "d_month", "d_year", "d_yearmonthnum", "d_yearmonth",
"d_daynuminweek", "d_daynuminmonth", "d_daynuminyear", "d_monthnuminyear",
"d_weeknuminyear", "d_sellingseason",
"d_lastdayinweekfl", "d_lastdayinmonthfl", "d_holidayfl", "d_weekdayfl",
"lo_orderkey", "lo_linenumber", "lo_custkey",
"lo_partkey", "lo_suppkey", "lo_orderdate", "lo_orderpriority",
"lo_shippriority", "lo_quantity", "lo_extendedprice",
```

```
"lo_ordertotalprice", "lo_discount", "lo_revenue", "lo_supplycost",
"lo_tax", "lo_commitdate", "lo_shipmode"  FROM "public".ORDERS
WHERE  ( STRPOS( "p_type", 'PROMO') > 0 )


REDSHIFT_49: Executed: on connection 0
Prepared statement REDSHIFT_48

NOTE: There were 4174 observations read from the data set MYRS.ORDERS.
      WHERE INDEX(p_type, 'PROMO')>0;
NOTE: The data set WORK.PROMOS has 4174 observations and 58 variables.
NOTE: DATA statement used (Total process time):
      real time           4.58 seconds
      cpu time            0.06 seconds
```

See “Passing SAS Functions to Amazon Redshift” in *SAS/ACCESS for Relational Databases* for more information about the SAS functions that can be translated into Redshift functions.

Implicit pass-through is a powerful optimization technique when you don’t know the SQL specifics of an accessed database in detail. It is also useful in tools such as SAS® Enterprise Guide® or SAS® Studio, because they mostly generate standard SAS SQL code.

Implicit pass-through is driven by these options, which are enabled by default:

- DIRECT_EXE | DBIDIRECTEXEC
- DIRECT_SQL
- SQL_FUNCTIONS

Explicit Pass-Through

Of course, you can still use explicit pass-through if you want to be in total control of the SQL code that is submitted to Redshift. This is done through the use of the SQL procedure with some special statements:

```
/* Explicit pass-through - explicit credentials - select data */
proc sql ;
   connect to redshift as myrs_pt(server="myredshift.amazonaws.com"
                                  database="mydb" user="myuser"
                                  password="XXXXXX") ;
   select * from connection to myrs_pt(
      select c_nation as customer_country, s_nation as supplier_country,
             sum(lo_quantity) as qty, sum(lo_revenue) as revenue
      from public.orders
      group by c_nation, s_nation ;
   ) ;
   disconnect from myrs_pt ;
quit ;

/* Explicit pass-through - implicit credentials - run specific Redshift
commands */
proc sql ;
   connect using myrs as myrs_pt ;
   execute (vacuum ;) by myrs_pt ;
   execute (analyze ;) by myrs_pt ;
   disconnect from myrs_pt ;
quit ;
```

To run a specific SQL string that returns a result set, you use the SELECT ... FROM CONNECTION TO ... syntax. Otherwise, if your SQL string does not produce any output, you use the EXECUTE statement.

Notice you have two ways to specify the database connection string. You can either specify it explicitly (`connect to redshift as`) or use an already existing library that contains the connection string (`connect using`).

Use FedSQL

Instead of the SQL procedure, you can use the FedSQL procedure to interact with Redshift. There might be benefits in using FedSQL instead of SQL (for example, ANSI SQL:1999 standard, data types, and so on.) See “Benefits of FedSQL” in *SAS® 9.4 FedSQL Language Reference*.

This is an example:

```
proc fedsql iptrace ;
   create table myrs.test_fedsql
      (
        key    TINYINT NOT NULL,
        field1 BIGINT NOT NULL
      ) ;
   insert into myrs.test_fedsql values (0,123456789012345) ;
   insert into myrs.test_fedsql values (1,123456789012345) ;
   insert into myrs.test_fedsql values (2,123456789012345) ;
quit ;
```

Notice that you cannot run this code using PROC SQL because it contains PROC SQL unsupported data types. But you can run it using explicit pass-through.

Run SAS Procedures In-Database

These common and useful Base SAS® procedures have been enhanced for in-database processing inside Amazon Redshift:

- FREQ
- RANK
- REPORT
- SORT
- SUMMARY/MEANS
- TABULATE

When you run these SAS procedures against Redshift data, most, if not all, of the complex processing is performed on the database side. This means that the SAS procedures are automatically converted into SQL code that runs close to the data, with no data movement. Only the result set, which is expected to be small, is downloaded into SAS for further processing and presentation on the client.

For example, running a TABULATE procedure on a huge Redshift table triggers an SQL summarization in Redshift. SAS is not involved during the initial computations. The result set, which should fit in a report table, is gathered in SAS for the presentation in the SAS user interface.

Some procedures can be totally offloaded in Redshift. If you run the following RANK procedure against a Redshift table and you want the output result to also be a Redshift

table, SAS/ACCESS to Redshift pushes down some SQL statements to make it happen without anything being computed on the SAS side.

Here is a partial extract from the SAS log:

```
82   proc rank data=myrs.orders out=myrs.ranks ;
83      var lo_revenue lo_supplycost ;
84      ranks rev_rank cost_rank ;
85   run ;

REDSHIFT_13: Executed: on connection 2
CREATE TABLE "public".RANKS (p_partkey NUMERIC(11),p_name
VARCHAR(22),p_mfgr VARCHAR(6),p_category VARCHAR(7),p_brand1
VARCHAR(9),p_color VARCHAR(11),p_type VARCHAR(25),p_size
NUMERIC(11),p_container VARCHAR(10),s_suppkey NUMERIC(11),s_name
...
 REDSHIFT: AUTOCOMMIT is YES for connection 2

 REDSHIFT_14: Executed: on connection 2
 INSERT INTO public.RANKS ( p_partkey, p_name, p_mfgr, p_category,
p_brand1, p_color, p_type, p_size, p_container, s_suppkey,
...
lo_revenue, lo_supplycost, lo_tax, lo_commitdate, lo_shipmode,
 rev_rank, cost_rank) SELECT "table0"."p_partkey" AS p_partkey,
"table0"."p_name" AS p_name, "table0"."p_mfgr" AS p_mfgr,
...
"table0"."lo_shipmode" AS lo_shipmode,
 "table1"."rankalias0" AS rev_rank, "table2"."rankalias1" AS cost_rank FROM
( SELECT "c_address" AS "c_address", "c_city" AS
 "c_city", "c_custkey" AS "c_custkey", "c_mktsegment" AS "c_mktsegment",
...
 "s_phone" AS "s_phone", "s_region" AS "s_region", "s_suppkey" AS
"s_suppkey" FROM public.ORDERS ) AS "table0" LEFT JOIN ( SELECT
 DISTINCT "lo_revenue", "tempcol0" AS "rankalias0" FROM ( SELECT
"lo_revenue", AVG( "tempcol1" ) OVER ( PARTITION BY "lo_revenue" )
...
"s_suppkey"
 AS "s_suppkey" FROM public.ORDERS ) AS "subquery7" WHERE ( (
"lo_supplycost" IS NOT NULL ) ) ) AS "subquery6" ) AS "subquery4" )
AS
 "table2" ON ( ( "table0"."lo_supplycost" = "table2"."lo_supplycost" ) )

 REDSHIFT: 25182 row(s) affected by INSERT/UPDATE/DELETE or other
statement.
 NOTE: PROCEDURE SQL used (Total process time):
       real time           0.63 seconds
       cpu time            0.06 seconds


NOTE: SQL generation was used to perform the ranking.
NOTE: PROCEDURE RANK used (Total process time):
      real time           0.77 seconds
      cpu time            0.14 seconds
```

EXTRACTING REDSHIFT DATA

In some situations, you might want to move the data from Redshift to SAS. There are many possible reasons to do this, such as

- you want to do something that cannot be done in SQL
- you have to move this data in an SAS in-memory engine
- you need to transform it before loading it in another third-party system
- it's part of an ETL process

Standard extract

If you are using SAS on AWS EC2 servers that are close to your Redshift instance, a standard extract (without any options) should be fast enough. It's the perfect scenario.

```
REDSHIFT_3: Prepared: on connection 0
SELECT * FROM "public".LINEORDER

82   data lineorder ;
83      set myrs.lineorder ;
84   run ;

REDSHIFT_4: Executed: on connection 0
Prepared statement REDSHIFT_3

NOTE: There were 14996590 observations read from the data set
MYRS.LINEORDER.
NOTE: The data set WORK.LINEORDER has 14996590 observations and 17
variables.
NOTE: DATA statement used (Total process time):
      real time           42.18 seconds
      cpu time            38.76 seconds
```

READBUFF

If you are using SAS on-premise, then you might want to check the READBUFF option, which specifies the number of rows of DBMS data to read into the buffer. It can help when there is some network latency between your SAS server and Redshift.

Here is a syntax example:

```
data lineorder ;
   set myrs.lineorder(readbuff=4096) ;
run ;
```

Bulk Unloading

Likewise, if you're still struggling with network latency between your SAS server and Redshift and you don't mind relying on AWS S3 to stage data temporarily, then the bulk unloading capability might be very useful (this capability is available starting with SAS 9.4M6).

The process is similar to bulk loading, but it works the other way around. Redshift first issues an UNLOAD statement to quickly move the data to S3 in a flat format. Then SAS downloads those files, parses them, and loads a local table.

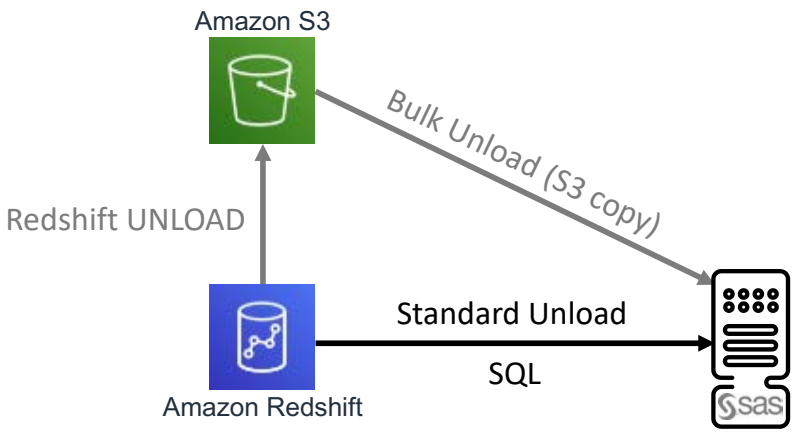

Figure 2 - Amazon Redshift to SAS Bulk Unloading Process

The code looks like this:

```
82   data lineorder ;
83      set myrs.lineorder(bulkunload=yes
84                         bl_bucket="mybucket/redshift_bulk_loading") ;
85   run ;

REDSHIFT_4: Executed: on connection 0
UNLOAD ('SELECT * FROM "public".LINEORDER') TO
's3://mybucket/redshift_bulk_loading/SASRSBL_176A0B35-6194-A248-8D77-
9F9746E28131_'
ACCESS_KEY_ID '' SECRET_ACCESS_KEY '' SESSION_TOKEN '' DELIMITER '\007'
MANIFEST

NOTE: There were 14996590 observations read from the data set
MYRS.LINEORDER.
NOTE: The data set WORK.LINEORDER has 14996590 observations and 17
variables.
NOTE: DATA statement used (Total process time):
      real time           1:16.30
      cpu time            37.79 seconds
```

WORK WITH REDSHIFT DATA FROM SAS VIYA

If you have already moved to SAS Viya, you will have to perform additional steps if you want to leverage your Redshift data in CAS, the SAS Viya in-memory analytics engine. The information in the first part of this paper still applies, because SAS Viya includes a SAS engine, which you will probably use to do the traditional Redshift data manipulations covered in previous topics in this paper.

We might use contradictory terms to describe loading and unloading operations. When we load data in CAS, we unload Redshift data, and vice versa. Hence, the use of Redshift bulk unloading options to load data in CAS, for example.

LOAD REDSHIFT DATA IN CAS

To manipulate, process, and analyze your Redshift data with SAS Viya capabilities, you first need to load it into CAS.

Standard loading

There are many programmatic ways in SAS Viya to load Redshift data into CAS, but the best way to start with is to use the CASUTIL procedure, which performs a server-side load.

```
/* Standard Redshift CASLIB */
caslib rs datasource=(srctype="redshift" server="myredshift.amazonaws.com"
                      database="mydb" schema="public"
                      user="myuser" password="XXXXXX") ;

proc casutil incaslib="rs" outcaslib="rs" ;
   load casdata="lineorder" casout="lineorder" ;
quit ;
```

Server-side loading means that the CAS cluster directly accesses the Redshift data source, the data goes directly from Redshift to the CAS controller, and the CAS controller distributes it to the CAS workers (if CAS is deployed in an MPP mode). Only the CAS controller must have the Redshift client (which is shipped with SAS Viya) configured in order to be able to connect to Redshift. This process is depicted in Figure 3.

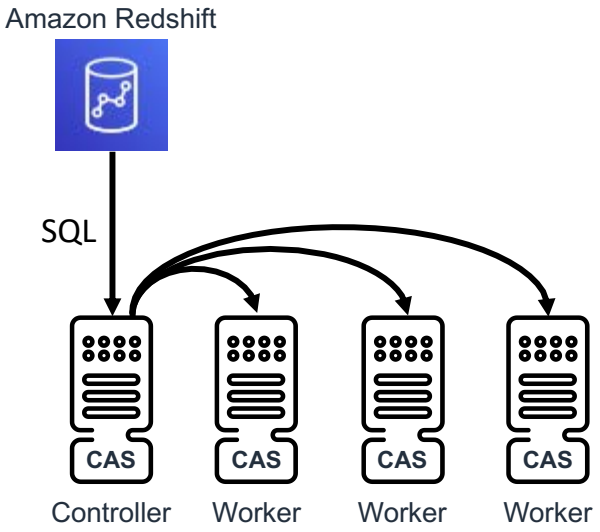

Figure 3 - Amazon Redshift to CAS Loading Process

READBUFF is still an option that can help in case of network latencies.

Multi-Node Loading

Multi-node loading is the process where CAS workers connect concurrently to the data source to load the data in parallel. The CAS controller coordinates the extracts. All of the CAS nodes must have the Redshift client configured in order to be able to connect directly to Redshift. CAS uses the SQL modulo function on the first suitable numeric column (if the user did not specify it using the SLICECOLUMN option) to instruct each CAS worker to query a portion of the data. Basically, in a three-worker configuration, each CAS worker queries and gets back one-third of the entire Redshift table (although it really depends on the numeric variable distribution).

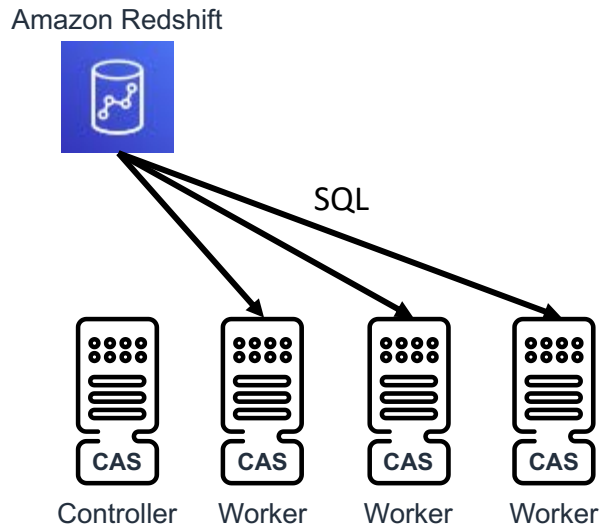

Figure 4 - Amazon Redshift to CAS Multi-Node Loading Process

Multi-node loading is driven by the CASLIB NUMREADNODES option, which specifies how many CAS nodes you want to use for querying the Redshift database in parallel.

```
caslib rs datasource=(srctype="redshift" server="myredshift.amazonaws.com"
                      database="mydb" schema="public" user="myuser"
                      password="XXXXXX" numreadnodes=10) ;

libname casrs cas caslib="rs" ;

proc casutil incaslib="rs" outcaslib="rs" ;
   load casdata="lineorder" casout="lineorder" ;
quit ;
```

NUMREADNODES can be set to these values:

- 0: use as many nodes as possible
- 1: use the CAS controller only (no multi-node mode)
- A value greater than 1: use up to the number of nodes indicated
    - NUMREADNODES=10 will use three nodes if your CAS cluster is made of three nodes
    - NUMREADNODES=10 will use 10 nodes if your CAS cluster is made of 20 nodes

These are typical SQL queries sent to Redshift when multi-node is activated (lo_orderkey is the first suitable numeric column):

Node 1:

```
select "SLICE_SQL"."lo_orderkey", ...
 from  (select "public"."lineorder"."lo_orderkey", ... from
        "public"."lineorder")  "SLICE_SQL"
 where  ( ( (MOD (ABS ("SLICE_SQL"."lo_orderkey") , 3) =0)  or
"SLICE_SQL"."lo_orderkey" IS NULL) )
```

Node 2:

```
select "SLICE_SQL"."lo_orderkey", ...
 from  (select "public"."lineorder"."lo_orderkey", ... from
        "public"."lineorder")  "SLICE_SQL"
```

```
  where  ( (MOD (ABS ("SLICE_SQL"."lo_orderkey") , 3) =1) )
```

Node 3:

```
  select "SLICE_SQL"."lo_orderkey", ...
   from  (select "public"."lineorder"."lo_orderkey", ... from
          "public"."lineorder")  "SLICE_SQL"
  where  ( (MOD (ABS ("SLICE_SQL"."lo_orderkey") , 3) =2) )
```

Bulk Unloading (Redshift) and Multi-Node Loading (CAS)

This method consists of bulk unloading (new in SAS Viya 3.5) the Redshift table to AWS S3 (using the Redshift UNLOAD command under the covers) and a parallel loading of the resulting file in CAS.

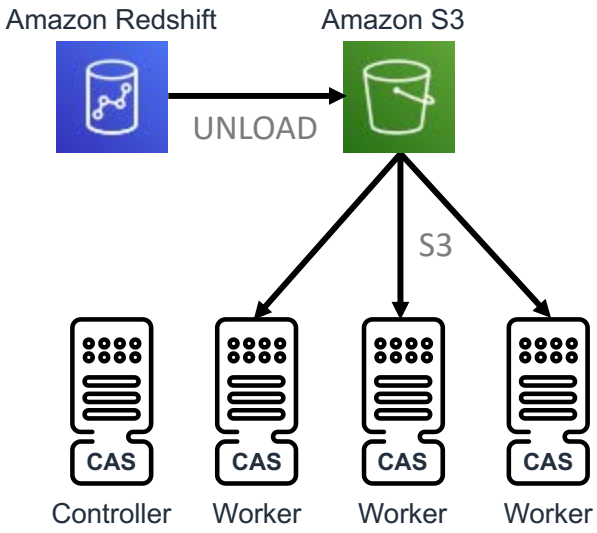

Figure 5 - Amazon Redshift to CAS Multi-Node Bulk Loading Process

Bulk unloading the Redshift table is only permitted when CAS MPP is used with multi-node loading enabled or a CAS SMP is used. It is not permitted when CAS MPP is used without multi-node loading enabled. The BULKUNLOAD options work together with the NUMREADNODES (value=0 or >1) option.

```
caslib rs datasource=(srctype="redshift" server="myredshift.amazonaws.com"
                      database="mydb" schema="public" user="myuser"
                      password="XXXXXX" numreadnodes=0) ;

proc casutil incaslib="rs" outcaslib="rs" ;
   load casdata="lineorder" casout="lineorder"
      datasourceoptions=(
         bulkunload=true
         awsconfig="/opt/sas/viya/config/data/AWSData/config"
         credentialsfile="/opt/sas/viya/config/data/AWSData/credentials"
         bucket="mybucket/redshift_bulk_loading"
      ) ;
quit ;
```

PROCESS REDSHIFT DATA FROM CAS

From a SAS Viya perspective, there is not much to do about processing Redshift data in place. SAS Viya is a data-crunching platform, and its in-memory engine must be fed with data coming from various data sources before you can exploit the SAS Viya capabilities. The

data loading phase is essential in a SAS Viya project. However, there are some SQL pass-through techniques that can be helpful when you want to load the result of an SQL query directly into CAS.

FedSQL Implicit Pass-Through Facility

FedSQL is the SAS SQL implementation that runs in CAS. It allows users to run their SQL queries in a scalable, threaded, and high-performance way. By default, it expects the input tables to be loaded in CAS before running the SQL query. However, if the input tables are not yet loaded in CAS, it is smart enough to automatically push the SQL query down to the database so that it is executed by the database and only the result set is loaded in CAS.

In this example, five tables are not loaded in CAS before running the FedSQL query, so the SQL query is completely offloaded to Redshift:

```
92   proc fedsql sessref=mysession _method ;
93      create table rs.orders as select
94      p.p_name, p.p_color, p.p_type, p.p_size,
95      s.s_nation, s.s_region,
96      c.c_nation, c.c_region,
97      d.d_month, d.d_year,
98      lo.lo_quantity, lo.lo_revenue
99      from rs.lineorder lo inner join rs.part p on
lo.lo_partkey=p.p_partkey
100        inner join rs.supplier s on lo.lo_suppkey=s.s_suppkey
101           inner join rs.customer c on lo.lo_custkey=c.c_custkey
102              inner join rs.dwdate d on lo.lo_orderdate=d.d_datekey
103     ;

Methods for full query plan
---------------------------
Number of Joins Performed is : 4
        MergeJoin (INNER)
          Sort
            MergeJoin (INNER)
              Sort
                MergeJoin (INNER)
                  Sort
                    MergeJoin (INNER)
                      Sort
                        SeqScan from RS.LINEORDER
                      Sort
                        SeqScan from RS.DWDATE
                  Sort
                    SeqScan from RS.SUPPLIER
              Sort
                SeqScan from RS.CUSTOMER
          Sort
            SeqScan from RS.PART

Offloaded SQL statement
-----------------------

        select "P"."p_name", "P"."p_color", "P"."p_type", "P"."p_size",
"S"."s_nation", "S"."s_region", "C"."c_nation",
"C"."c_region", "D"."d_month", "D"."d_year", "LO"."lo_quantity",
"LO"."lo_revenue" from "public"."LINEORDER" "LO" inner join
"public"."PART" "P" on  ("LO"."lo_partkey"="P"."p_partkey")  inner join
"public"."SUPPLIER" "S" on
```

```
("LO"."lo_suppkey"="S"."s_suppkey")  inner join "public"."CUSTOMER" "C" on
("LO"."lo_custkey"="C"."c_custkey")  inner join
"public"."DWDATE" "D" on  ("LO"."lo_orderdate"="D"."d_datekey")

        CAS output table: RS.ORDERS

NOTE: The SQL statement was fully offloaded to the underlying data source
via full pass-through
104  quit ;
NOTE: PROCEDURE FEDSQL used (Total process time):
      real time           0.79 seconds
      cpu time            0.01 seconds
```

If pass-through is not possible, the physical source tables are loaded temporarily into CAS on the fly and the request is processed in CAS (using load-on-demand capability)

To be eligible for implicit pass-through:

- All of the tables in the FedSQL request must exist in the same CASLIB.
- The tables cannot already have been loaded into CAS. If one of the tables is already loaded in CAS, the remaining tables are temporarily loaded into CAS on demand.

FedSQL Explicit Pass-Through Facility

Starting in SAS Viya 3.4, CAS and FedSQL introduced a new explicit pass-through facility. It allows to send native SQL SELECT-type syntax to a database for execution. The result set produced by the database is then loaded in CAS into memory for further processing.

Here is an example of a FedSQL explicit pass-through syntax:

```
proc fedsql sessref=mysession _method ;
   create table rs.extract as
   select * from connection to rs
      (
         select c_nation as customer_country, s_nation as supplier_country,
                sum(lo_quantity) as qty, sum(lo_revenue) as revenue
         from public.orders
         group by c_nation, s_nation
      ) ;
   ;
quit ;
```

SAVE CAS DATA TO REDSHIFT

In a SAS Viya data lifecycle, data is read and loaded into CAS, data is analyzed, resulting in new data sets, and possibly data is updated. Optionally, you might want to save your results back in Redshift. That's where saving CAS tables comes into play.

Standard Saving

Without any options specified, it is possible to save a CAS table directly to Redshift. As mentioned previously, Redshift inserts are slow. So it is best to avoid that method for writing data in Redshift.

A standard saving involves gathering the CAS data on the CAS controller before inserting it into a Redshift table.

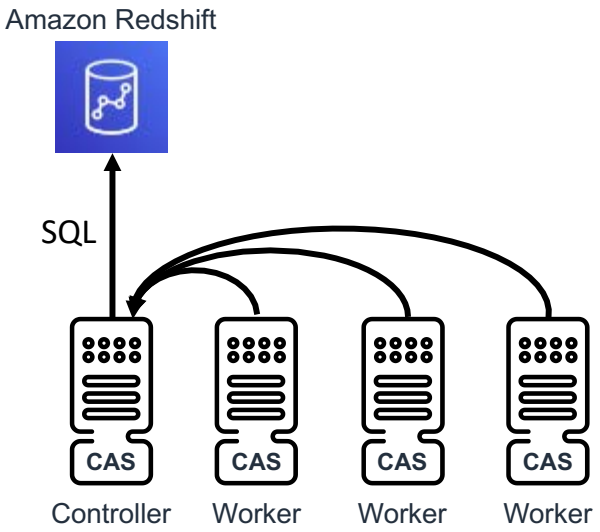

Figure 6 - CAS to Amazon Redshift Saving Process

Here is the syntax of a standard saving:

```
caslib rs datasource=(srctype="redshift" server="myredshift.amazonaws.com"
                      database="mydb" schema="public" user="myuser"
                      password="XXXXXX") ;
proc casutil incaslib="casuser" outcaslib="rs" ;
   save casdata="orders" casout="orders" replace ;
quit ;
```

Multi-Node Saving

Multi-node saving improves standard loading by making the CAS workers communicate directly with the database. Each node is responsible for inserting their local portion of data into Redshift.

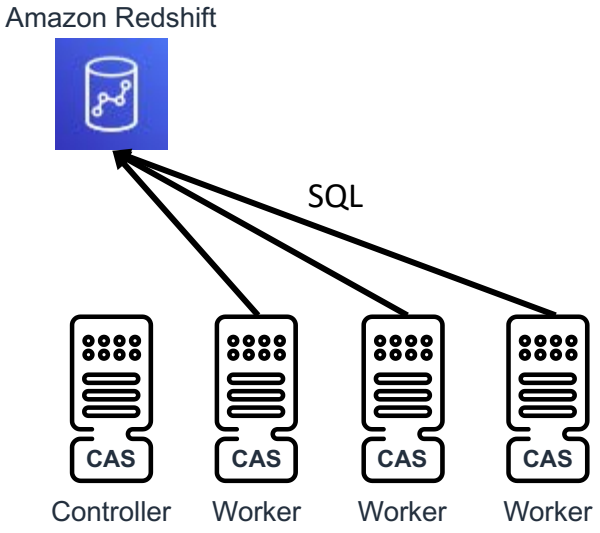

Figure 7 - CAS to Amazon Redshift Multi-Node Saving Process

Multi-node saving is driven by the CASLIB NUMWRITENODES option, which specifies how many CAS nodes you want to use in parallel for saving a CAS table into a Redshift database.

```
caslib rs datasource=(srctype="redshift" server="myredshift.amazonaws.com"
```

```
                        database="mydb" schema="public" user="myuser"
                        password="XXXXXX" numwritenodes=10) ;
proc casutil incaslib="casuser" outcaslib="rs" ;
   save casdata="orders" casout="orders" replace ;
quit ;
```

"Bulk Loading" Saving

Because of the poor Redshift insert performance, bulk loading in Redshift (available with SAS Viya 3.4) while saving a CAS table is the best solution to produce fast writing capabilities. Unlike the bulk unloading feature, which is supported only in MPP multi-node or in SMP, bulk loading can be done whether your CASLIB is defined with NUMWRITENODES=1 (the default value for saving from the CAS controller) or NUMWRITENODES=0 or >1 (multi-node saving).

Figure 8 illustrates the "single-node" scenario, where CAS data is gathered on the CAS controller, transferred to S3, and loaded into Redshift using the Redshift COPY statement.

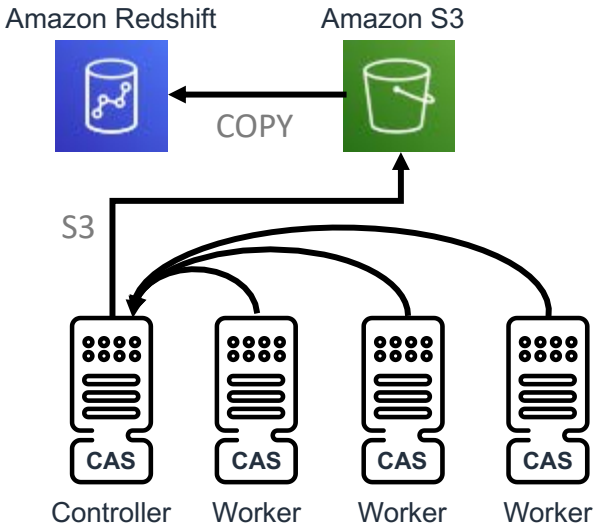

Figure 8 - CAS to Amazon Redshift Bulk-Saving Process

In terms of code, the options are pretty similar to those used for loading data in CAS. NUMWRITENODES=1 means that only the CAS controller will communicate with S3.

```
caslib rs datasource=(srctype="redshift" server="myredshift.amazonaws.com"
                        database="mydb" schema="public" user="myuser"
                        password="XXXXXX" numwritenodes=1) ;

proc casutil incaslib="rs" outcaslib="rs" ;
   save casdata="orders" casout="orders" replace
      options=(
         bulkload=true
         awsconfig="/opt/sas/viya/config/data/AWSData/config"
         credentialsfile="/opt/sas/viya/config/data/AWSData/credentials"
         bucket="mybucket/redshift_bulk_loading"
      ) ;
quit ;
```

Multi-Node Saving (CAS) and Bulk Loading (Redshift)

This is a variant of the previous scenario, where you want to take advantage of the multiple CAS workers to stage their portions of CAS data in parallel and directly into S3 before

loading the resulting files from S3 to Redshift using the COPY statement. This method skips the need for data gathering on the CAS controller.

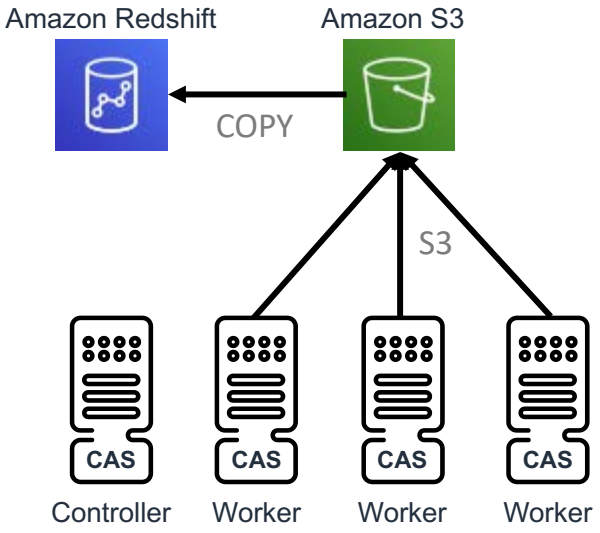

Figure 9 - CAS to Amazon Redshift Multi-Node Bulk-Saving Process

Compared to the previous example, only the NUMWRITENODES value is different. A value of 0 means all CAS workers will upload their data to S3.

```
caslib rs datasource=(srctype="redshift" server="myredshift.amazonaws.com"
                      database="mydb" schema="public" user="myuser"
                      password="XXXXXX" numwritenodes=0) ;

proc casutil incaslib="rs" outcaslib="rs" ;
   save casdata="orders" casout="orders" replace
      options=(
         bulkload=true
         awsconfig="/opt/sas/viya/config/data/AWSData/config"
         credentialsfile="/opt/sas/viya/config/data/AWSData/credentials"
         bucket="mybucket/redshift_bulk_loading"
      ) ;
quit ;
```

CONCLUSION

In release after release, SAS continuously improves existing capabilities and adds new features to better leverage Amazon Redshift data. Whether you are using SAS 9 or SAS Viya, it is crucial to access Redshift data quickly and efficiently, to process data where it makes sense, to inject Redshift data into the SAS in-memory analytics engine in a timely manner, and to save results back to Redshift. SAS currently provides a wide and mature variety of capabilities that help SAS customers to harness the potential of their Redshift data.

REFERENCES

SAS Institute Inc. 2019. “SAS/ACCESS Interface to Amazon Redshift.” In SAS/ACCESS® 9.4 for Relational Databases: Reference. 9th ed. Cary, NC: SAS Institute Inc. Available

https://documentation.sas.com/?cdcId=pgmsascdc&cdcVersion=9.4_3.5&docsetId=acreldb&docsetTarget=n16eh98vroso8jn11e5z8ztvb4ez.htm&locale=en

SAS Institute Inc. 2019. "Amazon Redshift Data Connector." In *SAS® Cloud Analytic Services 3.5: User's Guide*. Cary, NC: SAS Institute Inc. Available https://documentation.sas.com/?cdcId=pgmsascdc&cdcVersion=9.4_3.5&docsetId=casref&docsetTarget=p0kz8tkptx6206n1uu53y2gscaw6.htm&locale=en

SAS Institute Inc. 2019. "Benefits of FedSQL." In *SAS 9.4 FedSQL Language Reference*. 5th ed. Cary, NC: SAS Institute Inc. Available https://documentation.sas.com/?docsetId=fedsqlref&docsetTarget=p1hfs88bbog40gn1ir5z4v77pmca.htm&docsetVersion=9.4&locale=en

ACKNOWLEDGMENTS

Special thanks to Chris DeHart and Salman Maher for their help on this paper.

RECOMMENDED READING

- DeHart, Chris, and Bailey, Jeff. 2016 "It's raining data! Harnessing the Cloud with Amazon Redshift and SAS/ACCESS®.", Proceedings of the SAS Global Forum 2016 Conference. Cary, NC: SAS Institute Inc.Available http://support.sas.com/resources/papers/proceedings16/SAS5200-2016.pdf

CONTACT INFORMATION

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

Nicolas Robert
SAS Institute Inc.
nicolas.robert@sas.com
http://www.sas.com