Paper SAS4103-2020

An Insider's Guide to SAS/ACCESS® Interface to Snowflake

Jeff Bailey, SAS Institute Inc.

ABSTRACT

Snowflake is an exciting, new data warehouse built for the cloud. SAS/ACCESS® Interface to Snowflake allows SAS® to take advantage of this exciting technology. This paper describes Snowflake and details how it differs from other databases that you might have used in the past.

Using examples, we discuss the following topics:

- the differences between using SAS/ACCESS Interface to Snowflake and SAS/ACCESS® Interface to ODBC
- how to configure your SAS environment for SAS/ACCESS Interface to Snowflake
- tricks that you can use to discover what the SAS/ACCESS product is doing
- how to effectively move data into Snowflake using SAS/ACCESS Interface to Snowflake
- performance tuning your SAS and Snowflake environment

The paper uses an example-driven approach to explore these topics. Using the examples provided, you can apply what you learn from this paper to your environment.

INTRODUCTION

SAS released SAS/ACCESS Interface to Snowflake with SAS 9.4M6 and SAS Viya® 3.4; this new SAS/ACCESS product has proven very popular. This paper discusses what makes Snowflake different from other relational database management systems (RDBMS). We will look at these differences, explain why they matter, and show how you can use them with SAS to make your life easier.

As we discuss this new product, I will show examples and point out "the why" behind some of it.

Finally, I am going to do something I have never done before. This paper includes a section titled "Just Tell Me What to Do!" This new section lists best practices that will help you get great performance from the start. In short, it will make your first experience with SAS/ACCESS Interface to Snowflake a success. Yes, I realize that this is the only section of the paper that most people will read.

SAS/ACCESS Interface to Snowflake uses the Snowflake ODBC driver. Sometimes, this leads to confusion. This paper explores the many differences between SAS/ACCESS Interface to Snowflake and SAS/ACCESS Interface to ODBC and why you would choose one over the other.

AN INTRODUCTION TO SNOWFLAKE

Years ago, I was involved in a consulting project. I was the database administrator (DBA) on the team. My first task: get permission to purchase a UNIX machine, purchase said machine, find a place for the machine to live, arrange for a network connection, configure the operating system, install the database, and configure the database. Finally after all that, start working on the actual project. Exhausting!

This process took approximately 12 weeks. I spent a lot of this time waiting for other people to do "things." Frustrating is one way to describe the experience.

In the past, many projects followed this pattern. Spend a lot of money up-front. Order hardware. Wait. Install the software. Configure the software. Use the software. This pattern makes it difficult to spin-up projects quickly. Fortunately, now there is a solution.

Fast forward to modern times and things have changed. With Snowflake, you can go from no database to your first SQL query in a matter of minutes. All you need is a corporate email address and a credit card. It is simple.

Snowflake is a data warehouse created for the cloud and is not based upon an existing database, such as PostgreSQL.

Snowflake launched on Amazon Web Services (AWS). It is now also available for Microsoft Azure (Azure) and Google Cloud Platform (GCP). You can switch your cloud provider, and your database remains consistent – Outstanding! One of the many great things about Snowflake is that from an end-user perspective, it is just like those other SQL databases that you know and love.

There are a couple of things that make Snowflake different. Let's discuss my personal favorite.

COMPUTE AND STORAGE ARE SEPARATE

Traditional databases like Teradata and Oracle tie processing (compute) and managing data on disk (storage) together. If the database needs more disk space, a DBA can add it, but this is expensive and requires a great deal of planning or negotiating with other teams. In short, it can be expensive, painful, and difficult. Plus, do it too often, and people begin to question if you know what you are doing.

Compute is either the size of the machine running the database or, in the case of Teradata, the number of compute nodes. Increasing the compute resources allotted to a traditional database means either moving the database to a larger machine or buying new machines to add to the database. Both options are time-consuming and expensive.

The cost of increasing the capacity of a database is one reason DBAs tend to over-estimate the resources required for that new database system. The cost of being wrong is both expensive and embarrassing, not to mention time-consuming.

Snowflake makes this entire set of problems disappear by separating compute from storage. Let's discuss storage first because it is easier to explain. Snowflake uses the object store provided by the cloud it is running on. Cloud vendors say that their object stores (such as AWS S3) provide infinite capacity. I am sure there is a limit, but there is little chance of hitting it; this means that Snowflake DBAs will never face the issue of running out of space. Approaching management, with your tail between your legs, to ask for money for more disk space is a thing of the past.

Second, because compute and storage are separate, your DBA can automatically increase the resources allocated to compute. In Snowflake, this is called a warehouse. When your SAS® Visual Analytics queries begin to slow down, your DBA can increase the size of the warehouse for the environment. Magically, the queries speed up.

Have a huge bulk load job that you need to run? Your DBA can create a new Snowflake warehouse for it to use. Here is the cool part. The bulk load warehouse can load the SAS Visual Analytics warehouse while it is in use. That's right; multiple Snowflake warehouses can work on a single copy of the storage.

I am a DBA at heart. Here is what I like about Snowflake. When I create a Snowflake environment, I don't need to know exactly how much computer and storage I need; I start

small and increase as needed. In short, I am not buying four times the hardware I need just to be safe.

> Pro Tip: You can increase Snowflake performance by increasing the size of the Snowflake warehouse; this equates to adding machines, or switching to more powerful virtual machines (VM), to increase performance. Likewise, you can save money by reducing the capacity of a warehouse.

But that's not all. When the Snowflake environment is not used for a specified amount of time, it goes into an inactive state. This feature saves a lot of money because you are not paying for Snowflake when people are not using it.

SNOWFLAKE IS AVAILABLE ON MULTIPLE CLOUDS

Snowflake runs on AWS, Azure, and GCP. SAS initially developed SAS/ACCESS Interface to Snowflake on AWS. SAS supports general SQL functionality on all three clouds. Bulk loading requires interaction with the specific cloud object stores. At the time I am writing this paper, SAS supports bulk loading only on AWS. SAS intends to support bulk loading on Azure and GCP, but it is currently a work-in-progress. Check the SAS documentation to verify that SAS supports this bulk loading in your cloud environment.

SAS/ACCESS INTERFACE TO SNOWFLAKE VS. SAS/ACCESS INTERFACE TO ODBC

It is very common for people to ask me some form of this question, "What are the differences between SAS/ACCESS Interface to ODBC and SAS/ACCESS Interface to Whatchamacallit?" For our discussion, Whatchamacallit is Snowflake.

It's a great question, especially because SAS/ACCESS Interface to Snowflake uses the Snowflake ODBC driver. Let's take a look at the differences.

EXTENDED DATA TYPE INTEGRATION

SAS/ACCESS Interface to ODBC does not support all Snowflake-specific data types. Complicating matters is that the Snowflake ODBC driver returns values that deviate from the ODBC standard. This makes life more challenging for SAS/ACCESS Interface to ODBC users who use Snowflake. Customers have been calling SAS Technical Support (angels who have our backs) regarding the TIMESTAMP data types. The ODBC standard calls for a maximum precision of 29. However,Snowflake supports a maximum precision of 35 by default. Fortunately, SAS/ACCESS Interface to ODBC provides a means of handling this situation.

If you are using SAS/ACCESS Interface to ODBC and encounter a problem with the TIMESTAMP data type, you might find these commands and comments useful.

Snowflake recommends using the following Snowflake SQL command to adjust the data types:

```
ALTER SESSION SET ODBC_USE_CUSTOM_SQL_DATA_TYPES = true;
```

The following Snowflake SQL commands might help with TIMESTAMP columns:

```
ALTER SESSION SET TIMESTAMP_TYPE_MAPPING = TIMESTAMP_NTZ;
ALTER SESSION SET CLIENT_TIMESTAMP_TYPE_MAPPING = TIMESTAMP_NTZ;
```

If you need to set these options via SAS/ACCESS Interface to ODBC, the DBLIBINIT= LIBNAME option might help you.

If you are using SAS/ACCESS Interface to Snowflake, you don't have to worry about any of this; SAS takes care of these details for you.

EXTENDED SNOWFLAKE FUNCTION SUPPORT

The SAS/ACCESS Interface to Snowflake passes down more functions to the Snowflake server. Passing function calls to Snowflake can greatly enhance performance, especially when the function is included in the WHERE clause.

SAS passes the following functions to Snowflake for processing (See the SAS documentation for details.):

- ABS
- ARCOS (ACOS)
- ARSIN (ASIN)
- ATAN
- ATAN2
- CAT (CONCAT)
- CEIL
- COALESCE
- COS
- COSH
- COT
- DAY (DAYOFMONTH)
- DTEXTDAY (DAYOFMONTH)
- DTEXTMONTH (MONTH)
- DTEXTWEEKDAY (DAYOFWEEK)
- DTEXTYEAR (YEAR)
- EXP
- FLOOR
- HOUR
- INDEX (CHARINDEX)
- LEFT (LTRIM)
- LENGTH (OCTET_LENGTH(RTRIM()))
- LENGTHC (LENGTH)
- LOG (LN)
- LOG10 (LOG(10,n))
- LOG2 (LOG(2,n))
- LOWCASE (LOWER)
- MINUTE
- MOD
- MONTH
- QTR (QUARTER)
- REPEAT
- SECOND
- SIGN
- SIN
- SINH
- SQRT
- STD (STDDEV)
- STRIP (TRIM)
- SUBSTR
- TAN
- TANH
- TRANWRD (REGEXP_REPLACE)
- TRIMN (RTRIM)
- UPCASE (UPPER)
- VAR (VARIANCE)
- WEEKDAY (DAYOFWEEK)
- YEAR

SAS PROCEDURE PUSH-DOWN

SAS/ACCESS Interface to Snowflake pushes processing for the following Base SAS procedures inside Snowflake:

- FREQ
- REPORT
- SORT
- SUMMARY
- MEANS
- TABULATE

SAS/ACCESS Interface to ODBC does not push down SAS in-database procedures.

INTERNATIONALIZATION

The Snowflake ODBC driver supports internationlization (I18N). Unfortunately, SAS/ACCESS Interface to ODBC cannot make use of this capability. If your work requires I18N support, you need SAS/ACCESS Interface to Snowflake.

BULK LOADING

SAS/ACCESS Interface to Snowflake includes bulk loading for AWS. SAS/ACCESS Interface to ODBC does not support bulk loading. Bulk loading is by far the most common deciding factor when choosing between SAS ODBC and Snowflake products.

SNOWFLAKE VS ODBC: WHICH IS BEST?

For many of our customers, this is a very difficult question to answer. If Snowflake is one of many data sources you need to access and cost is an issue, SAS/ACCESS Interface to ODBC (and its JDBC counterpart) provide a degree of flexibility that is hard to beat. One product that empowers you to access data from hundreds of data sources has a lot going for it.

On the other hand, if your primary concern is Snowflake and you will be loading data into Snowflake your choice is clear – SAS/ACCESS Interface to Snowflake. The other benefits we discussed are icing on the cake.

DSN-LESS DATABASE CONNECTIONS

SAS/ACCESS Interface to Snowflake enables you to connect using SERVER= semantics. You do not have to configure or worry about having a Snowflake stanza in our SAS server's `odbc.ini` file. This makes life easier. Now is a great time to discuss connecting from SAS to Snowflake.

CONNECTING TO SNOWFLAKE

One of the many advantages of using SAS/ACCESS Interface to Snowflake over SAS/ACCESS Interface to ODBC is the simplified LIBNAME statement; the SAS Snowflake product has connection options for Snowflake.

Covering database connections is best done using examples.

Here is a simple SAS LIBNAME statement to connect to Snowflake:

```
libname mysnow snow server="mysnowflake.snowflakecomputing.com"
                    db=mydb
                    warehouse=mywh
```

```
                    schema=myuser
                    user=myuser
                    pw=mypassword
                    readbuff=32767
                    insertbuff=32767
                    dbcommit=0;
```

In this next example, the SAS CONOPTS= option enables you to pass along ODBC parameters when a SAS/ACCESS option is not available. This LIBNAME statement connects to Snowflake using a proxy server:

```
libname mysnow snow server="mysnowflake.snowflakecomputing.com"
db=mydb
warehouse=mywh
schema=myuser
user=myuser
pw=mypassword
readbuff=32767
insertbuff=32767
dbcommit=0
conopts="Proxy=my.gateway.mycompany.com:80;
ProxyUID=myuser;
ProxyPWD=myPassword;";
```

The CONOPTS= option is very sensitive to spaces. It is OK to include newlines in your code, but avoid spaces; this is why I formatted this code differently. CONOPTS= enables you to override parameter settings in the odbc.ini file. Overriding ODBC options in SAS code is especially useful in environments where you are not allowed to edit the odbc.ini file.

You are probably asking, "How can I find the list of the options that the ODBC driver supports?" The Snowflake documentation covers ODBC configuration and connection parameters. See the References section of this paper for details.

INSERTING DATA INTO SNOWFLAKE

Recommendation: Specify the READBUFF=, INSERTBUFF=, and DBCOMMIT= options for every Snowflake library that you create.

```
libname mysnow snow server="mysnowflake.snowflakecomputing.com"
                    db=mydb
                    warehouse=mywh
                    schema=myuser
                    user=myuser
                    pw=mypassword
                    readbuff=32767
                    insertbuff=32767
                    dbcommit=0;
```

This recommendation is important if you are using SAS/ACCESS Interface to Snowflake. It is a requirement if you are using SAS/ACCESS Interface to ODBC. These options can be set on the data sets too. The following code snippet shows how to set the INSERTBUFF= and DBCOMMIT= values for an INSERT operation:

```
proc append base=snow.insert_test_02 (insertbuff=32000 dbcommit=0)
            data=work.insert_test;
run;
```

When you read from Snowflake into SAS, be sure to include the READBUFF= option on either the LIBNAME statement or by using the READBUFF= data set option. The following example shows this option on a PROC SQL invocation:

```
proc sql;
   connect using mysnow;
   create table work.cars as
      select * from mysnow.cars (readbuff=3200);
run;
```

You are probably asking yourself, "Are INSERT statements the fastest way to get data into Snowflake?" The answer miraculously appears in the next section.

BULK LOADING DATA INTO SNOWFLAKE

Previously, I discussed the fact that Snowflake runs on multiple cloud platforms. However, you might have jumped to this section because you need to know how to use SAS/ACCESS Interface to Snowflake to bulk load data, and you don't have time for the other portions of this paper. With this in mind, I will repeat myself a little here. Snowflake uses its COPY command to load data from files that are available in cloud storage. If the file is being loaded from a machine that is not in the cloud, the file must be copied to a cloud storage location, called a stage, before it can be loaded into the target database table.

Because my SAS environment is on-premises, I use the BL_COMRESS= option to minimize the size of the load file.

Snowflake provides many options for setting the cloud storage location:

- User stages – Each Snowflake user has a stage allocated to them by default. User stages cannot be altered or dropped. This stage is for the user; do not share this with your friends. It is useful for doing non-production work. Don't use User stages for production work.
- Table stages – Each Snowflake table has a stage allocated to it by default. Table stages cannot be altered or dropped. Table stages are created when the table is created. These stages can be used for either ad hoc or production work, but are not appropriate if you need to load the data into multiple tables.
- Internal Named stages – Unlike user and table stages, internal named stages are created via the CREATE STAGE SQL command. If you are creating production data loads that involve multiple users or tables, this is the stage for you.

As we previously discussed, we can load data via an AWS S3 stage; this method bypasses Snowflake stages.

I found using Snowflake stages with SAS tricky. Fortunately, I have working examples and will explain the tricky parts to you. Let's take a look at it.

BULK LOADING INTO SNOWFLAKE USER STAGES

The following code successfully bulk loads a SAS data set into a new Snowflake table using a user stage:

```
/* Load via a Snowflake table stage */
data snow.large_table_1GB_table_stage
              (bulkload=yes
               bl_internal_stage="user/someuser"
               BL_COMPRESS=yes);
   set large_table_1GB;
run;
```

It may look strange, but "user/someuser" is exactly what I used when I ran this.

BULK LOADING INTO SNOWFLAKE TABLE STAGES

The following code successfully bulk loads a SAS data set into a new Snowflake table using a table stage:

```
/* Load via a Snowflake table stage */
data snow.large_table_1GB_table_stage
             (bulkload=yes
              bl_internal_stage="table/sometable"
              BL_COMPRESS=yes);
   set large_table_1GB;
run;
```

Yes, "table/sometable" is exactly what I used when I ran this.

BULK LOADING INTO SNOWFLAKE NAMED STAGES

The following code successfully bulk loads a SAS data set into a new Snowflake table using a pre-existing named internal stage:

```
/* Load via a Snowflake table stage */
data snow.large_table_1GB_table_stage
             (bulkload=yes
              bl_stage="MY_SNOW_STAGE"
              BL_COMPRESS=yes);
   set large_table_1GB;
run;
```

BULK LOADING INTO SNOWFLAKE VIA AN AWS S3 BUCKET

Snowflake runs on multiple cloud platforms. Each of these cloud platforms has its own variation of an object store. Think of an object store as a place to put a file in the cloud. It doesn't matter what kind of file. You can store a video, music, or a long-winded SAS Global Forum paper on an object store. At the time I am writing this (February 2020), the only object store supported for SAS/ACCESS Interface to Snowflake bulk loading is AWS S3. SAS intends to support Azure Data Lake Storage and Google Cloud Storage in the near future. Stay tuned.

Bulk loading via an external stage is more complicated than using one of the other Snowflake stage options because of the setup required.

To run an external bulk load using SAS/ACCESS to Snowflake, you need to complete these tasks:

- Create an S3 bucket with proper permissions. There are a variety of tutorials that you can use to create this bucket.
- Obtain the appropriate security keys and tokens.
- Determine your AWS connection profile. (This task is optional.)

I use the an S3 bucket named "snowbulk-bogus" in the following examples. To run an external bulk load using SAS/ACCESS to Snowflake, you will need to complete these tasks:

- Create an S3 bucket with proper permissions.
- Obtain the appropriate security keys and tokens.

- You may need to determine your AWS connection profile.

Let's assume you are running on Linux, have created an S3 bucket named "snowbulk-bogus," and have configured your AWS security information using the AWS recommended ~/.aws method. You can use the following DATA step to create a table and load data into it:

```
/* Load via a Snowflake external stage (AWS S3 bucket) */
data snow.large_table_1GB_s3_stage
             (bulkload=yes
              bl_bucket="snowbulk-bogus"
              bl_compress=yes);
   set large_table_1GB;
run;
```

If you have trouble getting this code to work, adding options could help you determine the source of your problem.

If you see the following error message, there is most likely a problem with your S3 bucket:

ERROR: Message from TKS3: HTTP/1.1 404 Not found

Perhaps you misspelled the bucket name; the bucket doesn't have proper privileges, or my personal favorite – you forgot to create it.

By default, AWS looks for the config and credentials files in the ~/.aws directory. Let's assume that your ~/.aws/credentials file contains a profile named "123456789012-bogus". Then here is your code:

```
data snow.large_table_1GB_s3_stage
                    (bulkload=yes
                     bl_bucket="snowbulk-bogus"
                     bl_aws_profile_name="123456789012-bogus"
                     bl_compress=yes);
   set large_table_1GB;
run;
```

There may be times when the config and credentials files are placed in a non-default location. The following code shows how you can tell SAS to look in a specific location for these files:

```
data snow.large_table_1GB_s3_stage
                  (bulkload=yes
                   bl_bucket="snowbulk-bogus"
                   bl_aws_profile_name="617292774228-production"
                   bl_aws_config_file="/prod/aws_secrets/config"
                   bl_aws_credentials="/prod/aws_secrets/credentials"
                   bl_compress=yes);
   set large_table_1GB;
run;
```

Specifying a non-default location for AWS security credentials is common for production ETL jobs. This type of work typically relies on automated mechanisms for generating AWS security tokens.

The previous approaches are fine when they work. But what should you do if you can't successfully run a SAS bulk load job? For instance, suppose you run your job and it fails with the following error message:

ERROR: Message from TKS3: The provided token is malformed or otherwise invalid.

This error message means that there is a problem with your credentials. The first thing you should do is regenerate the tokens and then rerun. If that doesn't work, there is a last resort.

The following example includes security tokens and keys; it is useful for debugging only. The primary benefit of this approach is that you can see what you are doing.

Do not do this in production code; it is best if you don't do this at all, but sometimes it makes debugging much easier. Never, ever, do this with permanent AWS keys; I am using temporary security credentials that expire in an hour. I assure you, these keys are now useless.

There is a cottage industry of crooks scouring GitHub looking for permanent keys. These crooks will use your keys (think of this as your credit card) and run expensive AWS resources at your expense. Even worse, these crooks can use your permanent keys to steal data.

Be very careful and use this example for debugging purposes only:

```
/* Take the unsecure route to see it work. */
/* BL_TOKEN= is AWS_SESSION_TOKEN= from the ~/.aws/credentials file */
/* DO NOT USE PERMANENT KEYS FOR THIS!!! */
data snow.large_table_1GB_s3_stage
       (bulkload=yes
        bl_bucket="snowbulk-bogus"
        bl_key="ASIAY9OMEHNKICF4K74B"
        bl_secret="mpSaAx9yVbJHHUfCf3CooabfueCfxBkR5aRDliiy"

bl_token="IQoJb3JpZ2luX2VjEBYaCXVzLWVhc3QtMSJHMEUCIQCsRsH7cCmM5Xt9TtKy9RT0T
mwkXlH6C+NUBg
qCvziuHgIgI4KxzqCQGWCvQ0fsgqLg9o0Us7wMr24Hvx1lAwDyHE8qpAIIrv/////////ARABG
gw2MTcyOTI3NzQyMjgiDLe0elcf
ccsh1bO09Cr4Ae5n2l8YQt96EQEGd/X8KkDMQQCr5KMeVjAxncthWF4kwnpRQgV+PbE0AIo1XBY
GfaNwqerTRIogalFQ43/+tyr6IZ
1ov8DTh+15/wCNULvtQHUWVx4VQtuu/JW219tBnr1gJXAu+Ixk6WQaGrfY22ibp02eGuZPrP2Zx
Rs5sqyD1ezPjfDpNe/oSZ8c28IW
sN6xf8afT6+UPHFkzVUgcWFTfqLP/fyREg3zkqfpyL42KeHIwJ+wShVvmIoZ00eqsUPKtPMp5+7
4Wh5CPMIysIalXGlASCP7Gk7ppq
Nya0k2J1iKYkIwPW30JgeRlP90yi/pce8fvVSkMKChvfEFOukB5/4VdoCOhbTTqSsCnd8kUlpbl
H6tnIxspFqLbKkuwkSBJqbwG9bh
+qDjLzbokZJGFXrlNG5ORGyf2jnFveOdy4mDgSrds6hh00BwaQqtvTd6257/eNqCvrPYyYyqZTo
bITiFNhYz4BRuW1OdWSHRYCEzto
QcRzfYyEry7St1jlQaMv2mXf1F1ARI/WcX25ZQslE3k0Gm8zaSCgKdh58CwOisNkQ0zFk1gdpIm
dWl2KFwaTA8gVkKjooGjIbemXfV
FHlbad0as8AI5ViP3jKu31bm5iUPx4pGswekULRsUMAfS2TvKDf7nuE="
        bl_compress=yes);
   set large_table_1GB;
run;
```

If you run similar code and it works, it means that there is a problem with your credentials or config file. Work backward to one of the secure means of running your code.

SAS BULK LOAD OPTIONS

The previous section showed some of these options in action. It is easy to get started by using very few options. But, and this is a huge "but," your success may lie with using a specific bulk-load option. It is a good idea to read the doc so that you have a passing familiarity with these options.

The following SAS bulk load LIBNAME statement options and data set options are available for Snowflake:

- BL_AWS_CONFIG_FILE=
- BL_AWS_PROFILE_NAME=
- BL_AWS_CREDENTIALS_FILE=
- BL_BUCKET=
- BL_COMPRESS=
- BL_CONFIG=
- BL_DEFAULT_DIR=
- BL_DELETE_DATAFILE=
- BL_DELIMITER=
- BL_ENCKEY=
- BL_INTERNAL_STAGE=
- BL_KEY=
- BL_NUM_DATAFILES=
- BL_NUM_READ_THREADS=
- BL_OPTIONS=
- BL_REGION=
- BL_SECRET=
- BL_TOKEN=
- BL_USE_ESCAPE=
- BL_USE_SSL=
- BULKLOAD=

See the latest SAS documentation for details because new options may be available.

I know what you are thinking, "How do I determine whether to use inserts or bulkload?" Let's take a look at this question.

JUST TELL ME WHAT TO DO!

Reading From Snowflake

When reading data from Snowflake using SAS, you should always start by setting the READBUFF= option to a number close to the maximum value (32767). I typically set READBUFF=32000 because it is close to the max and easy to remember. I have not seen a large performance difference between 32000 and 32767. Please understand, you might see a difference − so experiment.

> Pro Tip: Always set READBUFF= to a high number when you are starting. If you want to optimize read performance, experiment to find the best value for your situation. I set the READBUFF= option in the LIBNAME statement and use the data set option to override.

INSERT VS. BULK LOAD

With small data sets, there is not a huge benefit to using bulk loading. In fact, with small data sets, bulk loading might be slightly slower. I tend to argue that the difference is so small you might as well use bulk loading because the data set could grow large enough to cause problems with future INSERT statements.

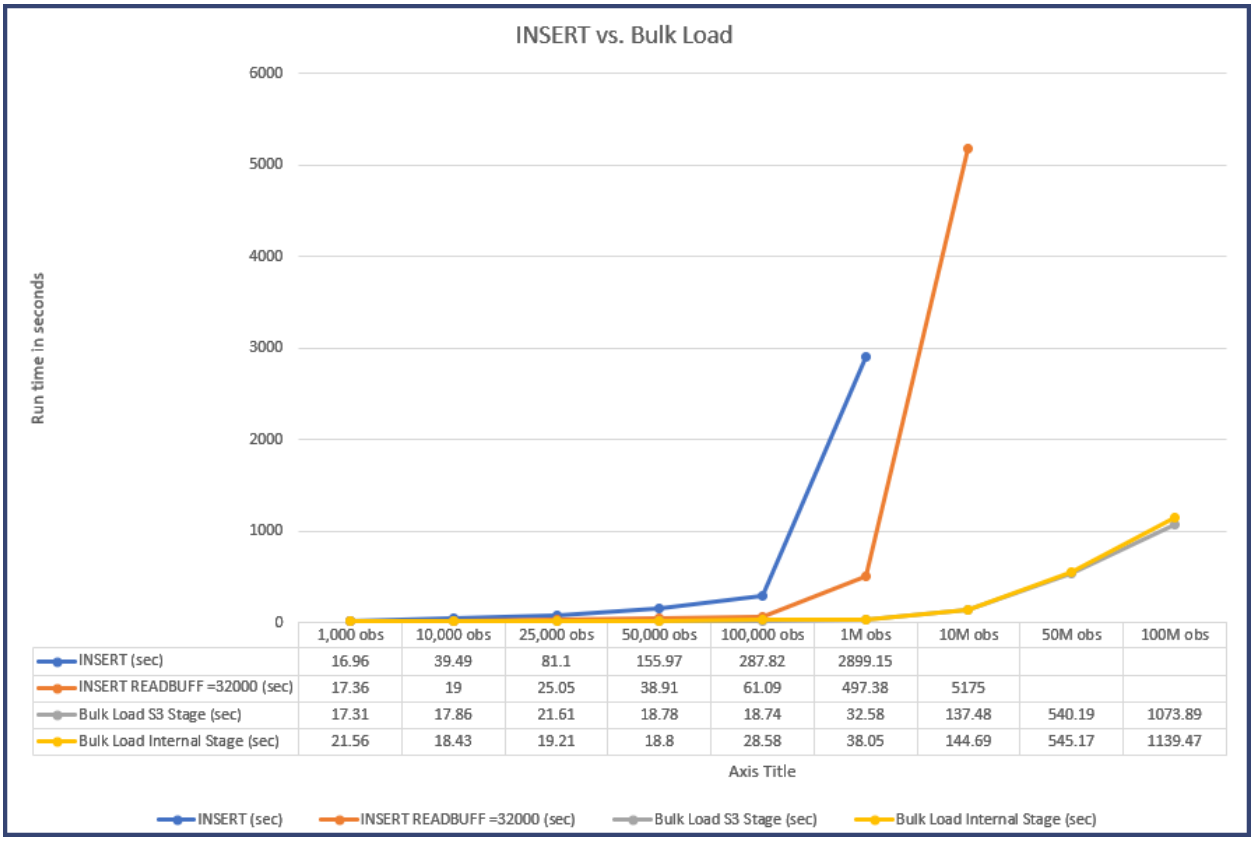
I have experimented with INSERT statements vs bulk load using an on-premises version of SAS Viya 3.5 running on a Linux machine. If you are running SAS in AWS, you can expect different numbers, but the exercise is valid.Use this exercise to determine the point at which bulk loading performs best.

There is no need to develop a complicated process to determine the performance characteristics of your environment; a simple approach works well. I used the following approach for this paper:

1. Find or generate a SAS data set for testing.
2. Run two INSERT jobs and check the performance. In both jobs, set DBCOMMIT=0 so that the new rows are committed at the end of the insert operation.
    a. In the first job, use the default value of INSERTBUFF=
    b. In the second job, set INSERTBUFF=32000 or some other large value
3. Run a bulk load and check the performance.
    a. Use an AWS S3 stage.
    b. Use a Snowflake internal stage. (A user stage or table stage is easiest.)
4. Determine where the cost versus benefit breakpoint is.

I graphed my test results and using the INSERTBUFF= option begins to beat simple INSERT statements somewhere between 1,000 and 10,000 observations. Bulk loading begins to shine somewhere between 25,000 and 50,000 observations.

Figure 1 shows the relative performance of various writing techniques.



	1,000 obs	10,000 obs	25,000 obs	50,000 obs	100,000 obs	1M obs	10M obs	50M obs	100M obs
INSERT (sec)	16.96	39.49	81.1	155.97	287.82	2899.15			
INSERT READBUFF =32000 (sec)	17.36	19	25.05	38.91	61.09	497.38	5175		
Bulk Load S3 Stage (sec)	17.31	17.86	21.61	18.78	18.74	32.58	137.48	540.19	1073.89
Bulk Load Internal Stage (sec)	21.56	18.43	19.21	18.8	28.58	38.05	144.69	545.17	1139.47

Figure 1. Comparison Graph of INSERT statements versus Bulk Load

> Pro Tip: Include the READBUFF=32000, INSERTBUFF=3200, and DBCOMMIT=0 options on every Snowflake LIBNAME statement you create. Feel free to experiment with the READBUFF= and INSERTBUFF= values, but always set DBCOMMIT=0.

Pro Tip: Use bulk loading by default.

By all means, experiment to find great values, but don't let the search for perfect stand in the way of good values.

HOW DO I MAKE MY QUERIES RUN FASTER?

This paper does not cover query tuning. Other papers (see "The SQL Tuning Checklist: Making Slow Queries a Thing of the Past" in the References section) cover this topic. One of the first steps in tuning a query is to find out what SAS is asking the database to do.

The following example shows how to use the OPTIONS statement in your SAS code:

```
libname mysnow snow server="mysnowflake.snowflakecomputing.com"
                    db=mydb
                    warehouse=mywh
                    schema=myuser
                    user=myuser
                    pw=mypassword
                    preserve_tab_names=no
                    readbuff=32767
                    insertbuff=32767
                    dbcommit=0;
data mysnow.cars;
   set sashelp.cars;
run;

options sastrace=',,,d' sastraceloc=saslog nostsuffix;

proc sql noexec;
   connect using mysnow;
   select count(*)
     from mysnow.cars
    where make in ('Toyota', 'Jeep');
quit;
```

Output 1 shows the results of running this code. The SQL that SAS submitted to Snowflake appears in bold. Once you have the SQL that SAS is sending to Snowflake, you can tune it. This example looks good because the count(*) function and the WHERE clause are both passed to Snowflake.

```
81   proc sql noexec;
82      connect using mysnow;
83      select count(*)
84        from mysnow.cars
85       where make in ('Toyota','Jeep');

SNOWFLAKE_1: Prepared: on connection 0
SELECT * FROM MYSNOW.CARS


SNOWFLAKE_2: Prepared: on connection 0
 select COUNT(*) from MYSNOW.CARS TXT_1 where ( TXT_1."Make" in ('Jeep',
'Toyota') )

NOTE: Statement not executed due to NOEXEC option.
86   quit;
NOTE: PROCEDURE SQL used (Total process time):
      real time           2.22 seconds
      cpu time            0.29 seconds

87
```

Output 1. Output from the SASTRACE= and SASTRACELOC= Options

CONCLUSION

SAS/ACCESS Interface to Snowflake is a very powerful product that makes using SAS with Snowflake easy. This paper has shown examples of how to connect to Snowflake using SAS LIBNAME statements and the CONNECT USING statement in the SQL procedure. We learned how to find the SQL that SAS is sending to Snowflake. Perhaps the most important lesson is when to use an INSERT statement and bulk loading.

We have only scratched the surface of what you can do with SAS/ACCESS Interface to Snowflake. Make sure that you read the SAS/ACCESS documentation so that you can get the most out of this product.

REFERENCES

Bailey, Jeff. 2017. "An Insider's Guide to Fine-Tuning Your CREATE TABLE statements using SAS® Options." *Proceedings of the SAS Global Forum 2017 Conference*. Cary, NC: SAS Institute Inc. Available https://support.sas.com/resources/papers/proceedings17/SAS0409-2017.pdf.

Bailey, Jeff. 2014. "An Insider's Guide to SAS/ACCESS® Interface to ODBC." *Proceedings of the SAS Global Forum 2014 Conference*. Cary, NC: SAS Institute Inc. Available https://support.sas.com/resources/papers/proceedings14/SAS039-2014.pdf.

Bailey, Jeff, and T. Petrova. 2013. "The SQL Tuning Checklist: Making Slow Database Queries a Thing of the Past." *Proceedings of the SAS Global Forum 2013 Conference*. Cary, NC: SAS Institute Inc. Available https://support.sas.com/resources/papers/proceedings13/080-2013.pdf.

Snowflake Inc. "ODBC Configuration and Connection Parameters." San Mateo, CA: Snowflake Inc. Available https://docs.snowflake.net/manuals/user-guide/odbc-parameters.html

ACKNOWLEDGMENTS

A handful of my favorite people were subjected to a litany of questions and asked to review the ideas presented in the paper. Thank you:

Carlos Bouloy, Snowflake

Chris DeHart, SAS Institute Inc.

Marie Dexter, SAS Institute Inc.

Keith Handlon, SAS Institute Inc.

Salman Maher, SAS Institute Inc.

RECOMMENDED READING

- *SAS® 9.4 and SAS Viya® 3.5 Programming Documentation*

CONTACT INFORMATION

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

Jeff Bailey
100 SAS Campus Drive
Cary, NC 27513
SAS Institute Inc.
Jeff.Bailey@sas.com
www.sas.com

APPENDIX

I used the following code to help me write this paper:

```
/************************/
/* PERFORMANCE TESTING */
/************************/

/* Selectively invoke tracing when needed. */
options sastrace=',,,d' sastraceloc=saslog nostsuffix;

/* Test the various Snowflake load techniques with largish data – 1GB */
/* Generate non-trivial data from performance experiments */
/* Change the DO loop to alter the data size */
/* 4.5M obs = 1 GB load file */

data work.large_table_1GB;
   format dt date.;

   do i = 1 to 4500000;
      dt = i + 1;
      mynum1 = i;
      text1  = "abcdefghijklmnopqrstuvwxyz0123456789";
      mynum2 = i * 10;
```

```
      text2  = "abcdefghijklmnopqrstuvwxyz0123456789";
      mynum3 = i * 10;
      text3  = "abcdefghijklmnopqrstuvwxyz0123456789";
      mynum4 = i * 10;
      text4  = "abcdefghijklmnopqrstuvwxyz0123456789";
      mynum5 = i * 10;
      text5  = "abcdefghijklmnopqrstuvwxyz0123456789";
      mynum6 = i * 10;
      output;
    end;
run;

libname snow SNOW server="saspartner.snowflakecomputing.com"
db=mydb
warehouse=mywh
schema=myuser
user=myuser
pw=mypassword
preserve_tab_names=no;

/* Get a baseline for INSERT performance */
data snow.large_table_1GB;
   set large_table_1GB;
run;

/* Load via a Snowflake internal named stage */
/* Based on the SAS doc, I am not sure how this one worked.     */
/* Must create an internal stage in Snowflake (via SQL) for this to work.
*/
data snow.large_table_1GB_named_stage (bulkload=yes
                                       bl_stage="JEFFS_STAGE"
                                       BL_COMPRESS=yes);
   set large_table_1GB;
run;

/* Load via a Snowflake table stage */
data snow.large_table_1GB_table_stage (bulkload=yes
bl_internal_stage="table/sometable" BL_COMPRESS=yes);
   set large_table_1GB;
run;

/* Load via a Snowflake user stage */
data snow.large_table_1GB_user_stage (bulkload=yes
                                      bl_internal_stage="user/someuser"
                                      BL_COMPRESS=yes);
   set large_table_1GB;
run;

/* Load via an S3 bucket */
/* Must create an S3 bucket for this to work */
/* Must ssh into the Linux machine where SAS is running */
/*    and obtain new credentials - kinit, getaswkey */
/* This, quite frankly, is a perfect world. It exists */
/*    to frustrate normal users and me. */
/* If [default] in your ~/.aws/credentials file is up to date, */
/*   it should work */
data snow.large_table_1GB_s3_stage (bulkload=yes
                                    bl_bucket="snowbulk-mybucket"
```

```
                                    bl_compress=yes);
   set large_table_1GB;
run;

/* Take the unsecure route to see it work. */
/* Only do this with temporary keys */
/* Do not use your permanent keys for this */
/* Do not do this in a production environment. */
/* BL_TOKEN= is AWS_SECURITY_TOKEN= from the ~/.aws/credentials file */

data snow.large_table_1GB_s3_stage (bulkload=yes
                                    bl_bucket="snowbulk-mybucket"
                                    bl_key="ASIAY7OMEXYKICF4K74B"

bl_secret="mpSaAx9yVbJHHUfCf3COiobfueCfxBkR5aRDliiy"

bl_token="IQoJb3JpZ2luX2VjEBYaCXVzLWVhc3QtMSJHMEUCIQCsRsH7cCmM5Xt9TtKy9RT0T
mwkXlH6C+NUBg
qCvziuHgIgI4KxzqCQGWCvQ0fsgqLg9o0Us7wMr24Hvx1lAwDyHE8qpAIIrv//////////ARABG
gw2MTcyOTI3NzQyMjgiDLe0elcf
ccsh1bO09Cr4Ae5n2l8YQt96EQEGd/X8KkDMQQCr5KMeVjAxncthWF4kwnpRQgV+PbE0AIo1XBY
GfaNwqerTRIogalFQ43/+tyr6IZ
1ov8DTh+15/wCNULvtQHUWVx4VQtuu/Jz21OtBnr1gJXAu+Ixk6WQaGrfY22ibp02eGuZPrP2Zx
Rs5sqyD1ezPjfDpNe/oSZ8c28IW
sN6xf8afT6+UPHFkzVUgcWFTfqLP/fyREg3zkqfpyL42KeHIwJ+wShVvmIoZ00eqsUPKtPMp5+7
4Wh5CPMIysIalXGlASCP7Gk7ppq
Nya0k2J1iKYkIwPW30JgeRlP90yi/pce8fvVSkMKChvfEFOukB5/4VdoCOhbTTqSsCnd8kUlpbl
H6tnIxspFqLbKkuwkSBJqbwG9bh
+qDjLzbokZJGFXrlNG5ORGyf2jnFveOdy4mDgSrds6hh00BwaQqtvTd6257/eNqCvrPYyYyqZTo
bITiFNhYz4BRuW1OdWSHRYCEzto
QcRzfYyEry7St1jlQaMv2mXf1F1ARI/WcX25ZQslE3k0Gm8zaSCgKdh58CwOisNkQ0zFk1gdpIm
dWl2KFwaTA8gVkKjooGjIbemXfV
FHlbad0as8AI5ViP3jKu31bm5iUPx4pGswekULRsUMAfS2TvKDf7nuE="
                                    bl_compress=yes);
   set large_table_1GB;
run;

/* I don't need the config and credential entries here, but I am showing
them as an example */
/* BL_AWS_PROFILE_NAME= is the important option here */
data snow.large_table_1GB_s3_stage (bulkload=yes
                                    bl_bucket="snowbulk-mybucket"
                                    bl_aws_profile_name="617292774228-
sandbox"
                                    bl_aws_config_file="~/.aws/config"
                                    bl_aws_credentials="~/.aws/credentials"
                                    bl_compress=yes);
   set large_table_1GB;
run;

/* Here is the minimum that you will likely need */
data snow.large_table_1GB_s3_stage (bulkload=yes
                                    bl_bucket="snowbulk-mybucket"
                                    bl_aws_profile_name="617292774228-
sandbox"
                                    bl_compress=yes);
   set large_table_1GB;
run;
```