SESUG Paper 130-2025

Simplifying SAS® Viya®: Explaining the Compute and CAS Servers, Caslibs, and In-Memory Data

Carleigh Jo Crabtree, SAS Institute Inc.

ABSTRACT

SAS Viya is a modern, cloud-enabled analytics platform designed for high-speed processing and scalability. It’s often described with terms like *in-memory* and *massively parallel processing*—but what do these concepts really mean in practice, and how can SAS 9 programmers continue to leverage their existing skills in this new environment?

This paper demystifies SAS Viya by breaking down the core capabilities of its two primary servers, explaining caslibs—what they are, how to use them, and how they compare to traditional SAS libraries—and demonstrating how to load data into memory for optimal performance. Helpful options and considerations when working with data on the CAS Server are also explored. Whether you’re new to Viya or transitioning from SAS 9, you’ll gain a clear, practical understanding of how to work effectively in this modern analytics platform.

INTRODUCTION

Moving from SAS 9 to SAS Viya can feel like stepping into a brand-new city—you recognize familiar landmarks, but the streets, traffic patterns, and even the language have changed. This paper serves as your “tour guide” to key SAS Viya concepts.

We begin with the architecture, breaking down the Compute and CAS Servers, and clarifying terms like single-threaded and multi-threaded processing. Next, we explore caslibs, comparing them to traditional SAS 9 libraries and highlighting the differences. We then demonstrate how to load data into memory, depending on its type and location. Finally, we cover helpful options when working with the CAS Server, and considerations for using the SUM statement and grouping data.

Throughout the paper, examples and results from program submissions are analyzed to give practical meaning to these concepts, helping you navigate SAS Viya with confidence.

THE COMPUTE AND CAS SERVERS

SAS Viya is a modern, cloud-enabled analytics platform that offers high-speed processing and scalability. It’s often described with terms like “in-memory,” “massively parallel processing,” and “cloud-native architecture”. Throughout this paper, I will explain and give meaning to these terms.

TRADITIONAL SAS 9: BACKGROUND

Traditionally, SAS was installed on a single system with dedicated disk, RAM, and CPUs. When the following program is run in SAS 9:

```
DATA work.cars_SUV;
     SET sashelp.cars;
     WHERE Type= "SUV";
RUN;


PROC MEANS data= work.cars_SUV;
     VAR MPG_Highway;
RUN;
```

```
PROC FREQ data= work.cars_SUV;
     TABLES Make;
RUN;
```

- The DATA step loads data from disk into memory, processes it, and writes it back to disk.
- The PROC MEANS step loads the data from disk again, generates results, and clears the memory.
- Finally, the PROC FREQ step also loads the data from disk, generates results, and clears the memory.

In this small program, the data is loaded and unloaded from memory three times, leading to repeated input/output (I/O) overhead. This is manageable for small programs but becomes time-consuming when working with large datasets or complex processes.

THE SAS VIYA DIFFERENCE: A TWO-SERVER MODEL

In SAS Viya, there are two primary servers, providing options to programmers depending on the processing demands of their programs. The two servers are:

1. SAS Compute Server
2. CAS (Cloud Analytic Services) Server

SAS Compute Server

The Compute Server works similarly to the traditional SAS 9 Server, which could have been referred to as PC SAS or the SAS Workspace Server. You can run existing SAS programs with normal SAS code in SAS Viya and it will execute on the Compute Server. It's a great option for smaller datasets or if you want to transition existing code into SAS Viya with minimal modifications. On the Compute Server, data is read into memory and cleared from memory with each step. Some processes are single-threaded (completing one task at a time), others are multi-threaded (tasks are being divided and worked on at the same time).

Threads Explained

CPUs have cores which are the physical processing units that execute instructions independently. Threads are the virtual sequences of instructions that can run on a single core.

Single-threaded processes execute instructions one at a time. This can be less efficient for tasks that can be parallelized or require waiting for I/O operations.

Multi-threaded processes allow multiple parts of a program to run concurrently. Different threads can execute different tasks simultaneously, which is more efficient. Multi-threading allows memory and files to be shared, and it can remain responsive even if part of it is blocked or performing a lengthy operation.

CAS Server

The CAS Server is fully multi-threaded, which is what allows programs to run faster, as tasks are being distributed.

The real power of SAS Viya comes from the CAS Server, which offers several advantages over traditional SAS processing:

- **Massively Parallel Processing (MPP) Environment**: CAS divides large datasets into chunks and processes them across multiple nodes simultaneously. This is like having several security lanes open at an airport, speeding up the entire process.
- **In-Memory Analytics Engine**: Instead of loading and unloading data multiple times like in SAS 9, CAS allows you to load a dataset into memory once and keep it there until your work is finished. This greatly reduces I/O, making processing faster and more efficient.

Why Does Server Choice Matter in SAS Viya?

Understanding the SAS Viya's servers is key to knowing when and how to leverage the capabilities of each processing environment. The CAS Server shines when you're working with:

- Large datasets (over 50GB)
- Programs that require multiple reads of the same data
- Long-running or computationally intensive tasks

By understanding when to use the Compute Server versus the CAS Server, you can optimize your code for better performance.

SAS Viya is deployed to a Kubernetes cluster. Typically, this is something only an admin needs to know about. However, as we learn how the CAS Server works, we refer to a controller and worker nodes, which are part of a Kubernetes cluster. Very simplified, a Kubernetes cluster is a set of node machines for running containerized applications. It consists of a control plane which is the "brain" of the cluster and nodes which are the worker machines. Among other things, the control plane or "controller node" has the ability to send tasks to the worker nodes.

HOW CAS WORKS

The CAS Server is configured to run on one or more machines, each with multiple nodes. Typically, there is one controller node and several worker nodes. When using SAS Studio in SAS Viya, the Compute Server acts as the client to the CAS Server. For data to be processed in CAS, it must be loaded into memory from a physical location. Caslibs connect to data source files and load tables into memory for processing in CAS.

When a program is executed in CAS, the controller node distributes in-memory data in blocks to the multiple worker nodes. The workers execute the same actions at the same time on different blocks of data, which is called parallel processing. As the worker nodes finish processing, they send the data back to the controller, and we see the results back in SAS Studio as expected.

Data that is in memory will stay in memory until it is explicitly dropped or the CAS session ends. While data is in memory, the same table can be accessed by multiple users, avoiding additional I/O. If you want to save changes made to in-memory tables, you must explicitly save them back to a physical storage location.

CASLIBS

Caslibs are the mechanism for accessing data in CAS. The focus of this section is what caslibs are, how to use them and how they compare to traditional SAS libraries.

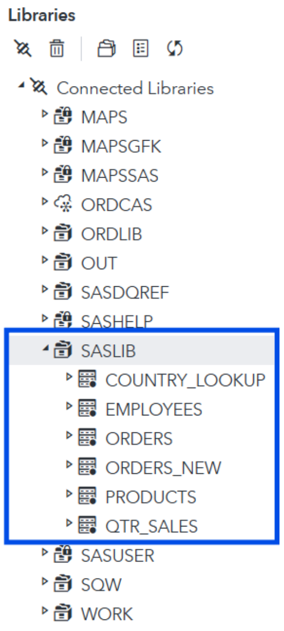
TRADITIONAL SAS LIBRARIES

In SAS 9, libraries are created with the LIBNAME statement and connect to data sources such as databases, cloud data, or folder paths. A library reference or "*libref*", acts as an alias to the physical data source. In code, data is referenced as *libref.tablename*. Libraries function similarly in SAS Viya when working with data on the Compute Server.

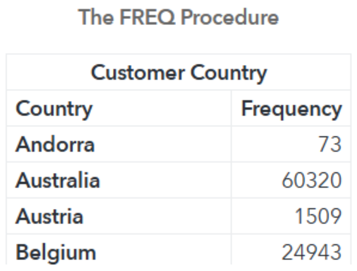
Below is an example of creating a libref named **sasLib** for SAS tables in the **data** folder. Using the FREQ procedure, the **orders** table in **sasLib** was used to display the number of orders placed in each country.

```
LIBNAME sasLib base
"/export/viya/homes/carleighjo.crabtree@sas.com/Courses/PGVY/data";


PROC FREQ data=sasLib.orders;

     TABLES Country /nocum nopercent;

RUN;
```

The library **sasLib** is visible in the Libraries pane, containing six SAS tables.



Partial results:



The FREQ Procedure

Customer Country	
Country	Frequency
Andorra	73
Australia	60320
Austria	1509
Belgium	24943

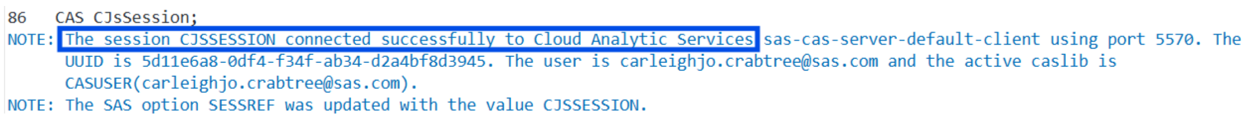
This code executes on the Compute Server. Running code on the Compute Server works well unless you're working with a large dataset (over 50GB), programs that requiring multiple reads of the same data, or computationally intensive tasks. In those cases, using the CAS Server is beneficial due to its high speed, in-memory processing capabilities.

WORKING WITH DATA IN CAS

What changes when working with data on the CAS server? One of the benefits of working in CAS is that data is held in-memory, reducing I/O. To process data in-memory, we must first create a CAS session. The session encapsulates or “keeps a record” of our work and connects us to the CAS Server. Use a CAS statement to create a session. For example:

```
CAS CJsSession;
```

Partial log:



```
86   CAS CJsSession;
NOTE: The session CJSSESSION connected successfully to Cloud Analytic Services sas-cas-server-default-client using port 5570. The
      UUID is 5d11e6a8-0df4-f34f-ab34-d2a4bf8d3945. The user is carleighjo.crabtree@sas.com and the active caslib is
      CASUSER(carleighjo.crabtree@sas.com).
NOTE: The SAS option SESSREF was updated with the value CJSSESSION.
```

Now we’re connected to the CAS Server, the session is named **CJsSession** and the CAS Server is ready to process in-memory data. After establishing a session, connect data to the CAS Server using caslibs. Caslibs connect to a variety of data sources such as data in the cloud, databases, folder paths, and streaming data.

Creating a Caslib

In SAS Viya, predefined caslibs are available, and users can create their own caslibs with permission from the SAS administrator. Use the CASLIB statement to create a caslib:

```
CASLIB caslibName PATH="/filepath/" LIBREF=libref;
```

- **caslibName**: Up to 256 characters, cannot start with a number, and contains only numbers, letters and underscores.
- **PATH=** option: Specifies the location of the physical data files. Unlike traditional SAS libraries, caslibs can contain multiple file types.
- **LIBREF=** option: Maps a libref to the caslib. The libref must follow standard naming conventions. It's common to make the libref and caslib names the same for clarity.

Why Map a Libref to a Caslib?

Mapping a libref to a caslib serves two critical purposes:

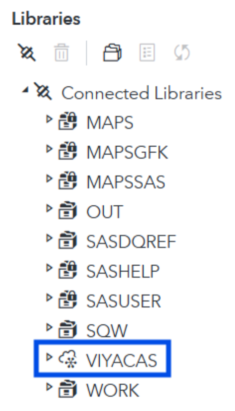
1. The mapped libref is visible in the Libraries pane, showing the caslib and any available in-memory tables.
2. The libref allows referencing in-memory tables in traditional SAS DATA steps and procedures as *libref.tablename*.

For example, let's create a caslib for the data source files in the **data** folder. Without the LIBREF= option, the caslib **dataCaslib** is created successfully but it is not visible in the Libraries pane.

```
CASLIB dataCaslib
PATH="/export/viya/homes/carleighjo.crabtree@sas.com/Courses/PGVY/data";
```

Mapping the LIBREF= option displays the libref connected to the caslib in the Libraries pane. You can recognize it is a libref mapped to a caslib by its cloud icon rather than the file cabinet icon. The example below creates the **viyaCas** caslib and assigns the libref **viyaCas**.

```
CASLIB viyacas
PATH="/export/viya/homes/carleighjo.crabtree@sas.com/Courses/PGVY/data"
LIBREF=viyacas;
```



Partial log:

```
82   CASLIB viyacas PATH="/export/viya/homes/carleighjo.crabtree@sas.com/Courses/PGVY/data" LIBREF=viyacas;
NOTE: 'VIYACAS' is now the active caslib.
NOTE: Cloud Analytic Services added the caslib 'VIYACAS'.
NOTE: CASLIB VIYACAS for session CJSSESSION will be mapped to SAS Library VIYACAS.
```

Mapping a Libref to a Predefined Caslib

Predefined caslibs are often set up by a SAS administrator. To use a predefined caslib that is not visible in the Libraries pane, map a libref to the caslib with the LIBNAME statement.

```
LIBNAME libref CAS CASLIB=caslibName;
```

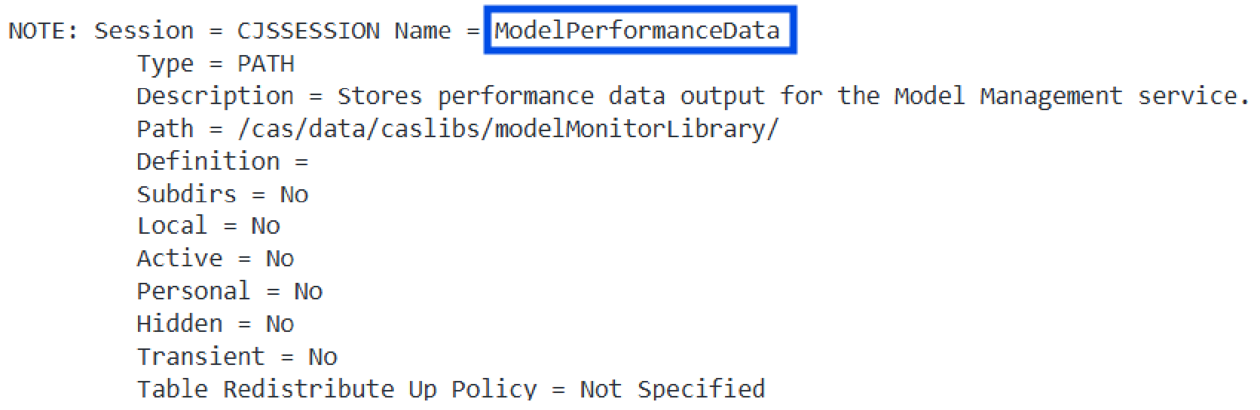
- **Libref**: Must follow SAS naming conventions and should be the same or representative of the **caslibName**.
- **Engine**: Always **CAS** for caslibs.
- **CASLIB=** option: Specifies the existing caslib. Caslib names can be longer than librefs, and may contain spaces. If the **caslibName** contains a space, add quotation marks around it:

  ```
  LIBNAME libref CAS CASLIB= 'caslib Name With Spaces';
  ```

To view all predefined caslibs, run:

```
CASLIB _all_ list;
```

One predefined caslib in my environment is **ModelPerformanceData**.

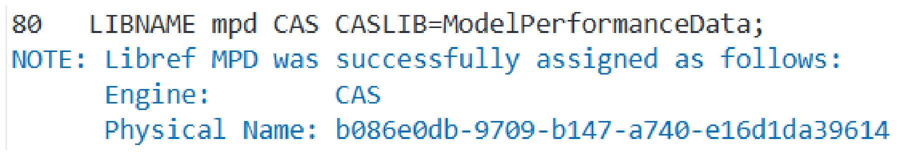


```
NOTE: Session = CJSSESSION Name = ModelPerformanceData
         Type = PATH
         Description = Stores performance data output for the Model Management service.
         Path = /cas/data/caslibs/modelMonitorLibrary/
         Definition =
         Subdirs = No
         Local = No
         Active = No
         Personal = No
         Hidden = No
         Transient = No
         Table Redistribute Up Policy = Not Specified
```

To create a libref named **mpd** to that caslib, run:

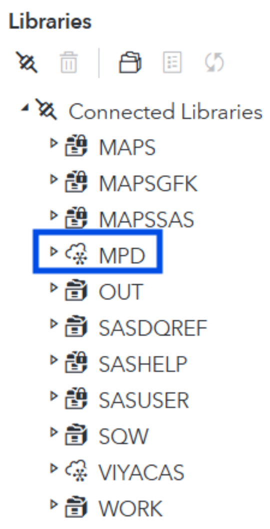
```
LIBNAME mpd CAS CASLIB=ModelPerformanceData;
```

Partial Log:



```
      LIBNAME mpd CAS CASLIB=ModelPerformanceData;
NOTE: Libref MPD was successfully assigned as follows:
      Engine:        CAS
      Physical Name: b086e0db-9709-b147-a740-e16d1da39614
```

The libref **mpd** mapped to the caslib **ModelPerformanceData** now appears in the libraries pane:



Caslib Attributes

Caslibs have attributes that describe their data connection and user access. The three main attributes are Local, Active and Personal.

Local

- **Local=Yes**: A local caslib is "session scope". If the caslib is created in SAS Studio, it is not visible in another application like SAS Visual Analytics. When the CAS session ends, the caslib is deleted. Session scope is useful when working with data that is not shared across sessions.
    - Traditional SAS comparison: Traditional SAS libraries are deleted at the end of a SAS session. You must run a LIBNAME statement to work with your data again.
- **Local=No**: If a caslib is not local, it is "global scope". The caslib is available to anyone with permission to access it, and the caslib is visible across applications. When the CAS session ends, the caslib is not deleted. Global scope is useful when sharing data across sessions, with other users and when working with large data that you do not want to load and unload from memory often.
    - Traditional SAS comparison: Traditional SAS has predefined libraries such as SASHELP that persist across SAS sessions.

Personal

- **Personal=Yes**: The caslib is only available to you.
- **Personal=No**: The caslib is available to other users.

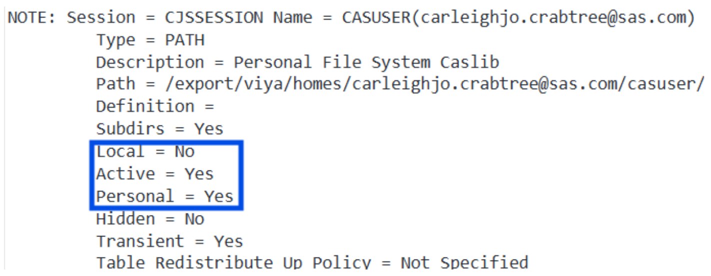
Active

In traditional SAS, the **work** library is the default when a library is not specified in a program. The CAS version of the **work** library is the "active" caslib. The active caslib can change. It is typically **casuser** by default, however, if you have just created a caslib, that new caslib will become the active caslib. It is best practice to specify the caslib you are working with.

- **Active=Yes**: The caslib is the current default caslib if no other caslib is specified.
- **Active=No**: The caslib is available to use but the caslib name must be specified to use it.

To view the **casuser** caslib attributes, use the CASLIB statement:

```
CASLIB casuser list;
```

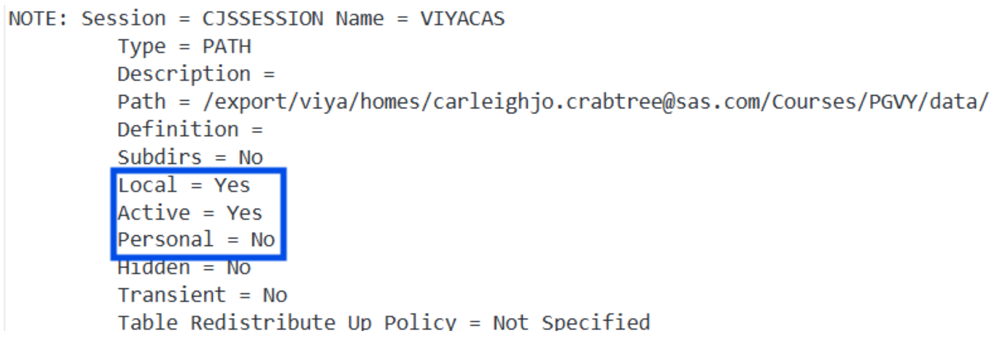


```
NOTE: Session = CJSSESSION Name = CASUSER(carleighjo.crabtree@sas.com)
        Type = PATH
        Description = Personal File System Caslib
        Path = /export/viya/homes/carleighjo.crabtree@sas.com/casuser/
        Definition =
        Subdirs = Yes
        Local = No
        Active = Yes
        Personal = Yes
        Hidden = No
        Transient = Yes
        Table Redistribute Up Policy = Not Specified
```

Casuser is a global scope caslib, it is currently the active/ default caslib if no other caslib is specified and it is personal meaning it is available only to me.

Creating the new caslib **viyaCas** will change the active caslib to **viyaCas**. It is also session scope, and it is not a personal caslib.

```
CASLIB viyacas
PATH="/export/viya/homes/carleighjo.crabtree@sas.com/Courses/PGVY/data"
LIBREF=viyacas;


CASLIB viyaCas list;
```

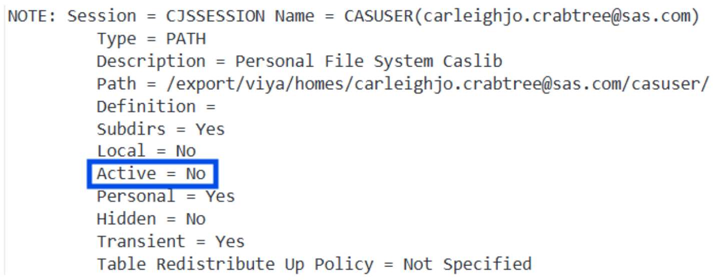


```
NOTE: Session = CJSSESSION Name = VIYACAS
        Type = PATH
        Description =
        Path = /export/viya/homes/carleighjo.crabtree@sas.com/Courses/PGVY/data/
        Definition =
        Subdirs = No
        Local = Yes
        Active = Yes
        Personal = No
        Hidden = No
        Transient = No
        Table Redistribute Up Policy = Not Specified
```

Run the following again:

```
CASLIB casuser list;
```

Notice that Active = No because **viyaCAS** is the active caslib.



```
NOTE: Session = CJSSESSION Name = CASUSER(carleighjo.crabtree@sas.com)
        Type = PATH
        Description = Personal File System Caslib
        Path = /export/viya/homes/carleighjo.crabtree@sas.com/casuser/
        Definition =
        Subdirs = Yes
        Local = No
        Active = No
        Personal = Yes
        Hidden = No
        Transient = Yes
        Table Redistribute Up Policy = Not Specified
```

Files vs. Tables in Caslibs

Caslibs can connect to a variety of data sources such as data in the cloud, databases, folder paths and streaming data.

Traditional SAS libraries have one main component, the connection to the data source file. Caslibs have three main components:

1. The connection to the data source files.
2. The in-memory portion for data that has been loaded into memory.

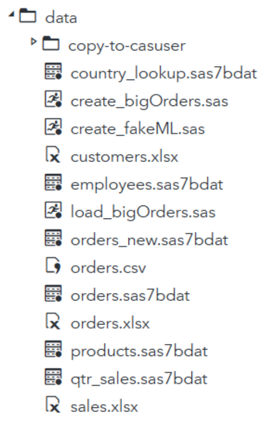
3. Access controls to define permissions to a specific caslib.

Use the CASUTIL procedure to view data source files and in-memory tables in a caslib:

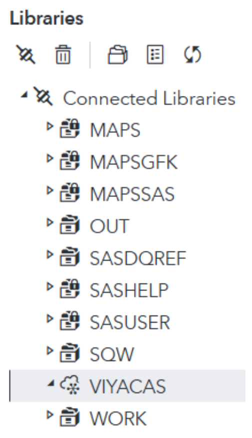
```
PROC CASUTIL <INCASLIB="caslib-name">;
     LIST FILES | TABLES;
QUIT;
```

INCASLIB= is optional, however if a caslib is not specified, the active caslib will be assumed. This is one instance where I recommend being explicit with the caslib name since the active caslib can change.

Let's look at an example. Previously, we mapped a libref to the caslib **viyaCas**. The folder **data** has several files, and a variety of file types.



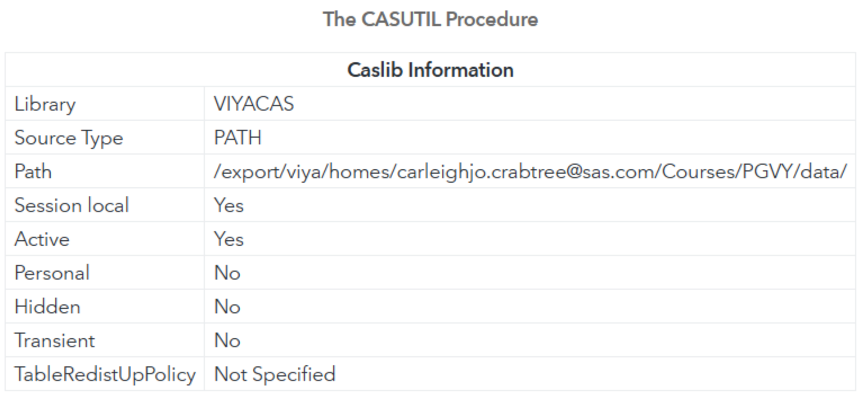
When looking at the **viyaCas** libref in the Libraries pane, the library appears empty:



This is because only in-memory tables are visible in a library mapped to a caslib in the Libraries pane. The following step displays the data source files available in the **ordCas** caslib:

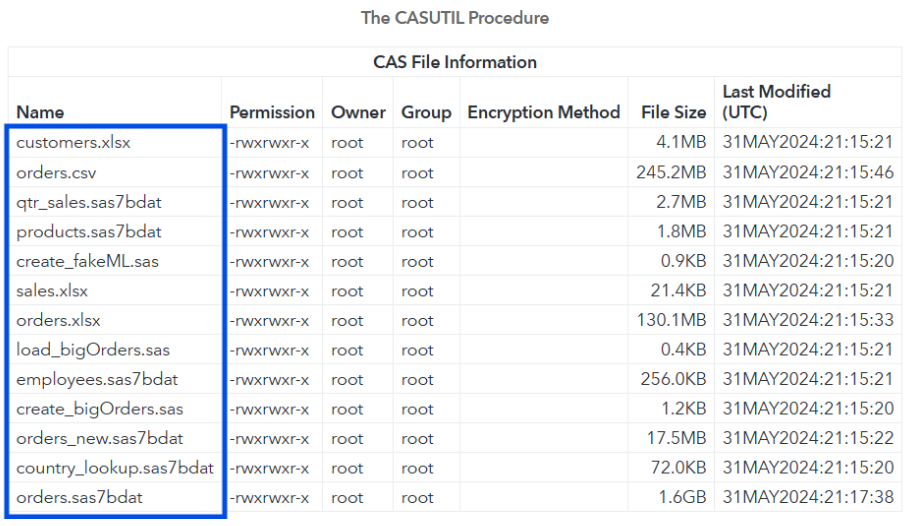
```
PROC CASUTIL INCASLIB="viyaCas";
     LIST files;
QUIT;
```

Results:



The CASUTIL Procedure

Caslib Information	
Library	VIYACAS
Source Type	PATH
Path	/export/viya/homes/carleighjo.crabtree@sas.com/Courses/PGVY/data/
Session local	Yes
Active	Yes
Personal	No
Hidden	No
Transient	No
TableRedistUpPolicy	Not Specified



The CASUTIL Procedure

CAS File Information

Name	Permission	Owner	Group	Encryption Method	File Size	Last Modified (UTC)
customers.xlsx	-rwxrwxr-x	root	root		4.1MB	31MAY2024:21:15:21
orders.csv	-rwxrwxr-x	root	root		245.2MB	31MAY2024:21:15:46
qtr_sales.sas7bdat	-rwxrwxr-x	root	root		2.7MB	31MAY2024:21:15:21
products.sas7bdat	-rwxrwxr-x	root	root		1.8MB	31MAY2024:21:15:21
create_fakeML.sas	-rwxrwxr-x	root	root		0.9KB	31MAY2024:21:15:20
sales.xlsx	-rwxrwxr-x	root	root		21.4KB	31MAY2024:21:15:21
orders.xlsx	-rwxrwxr-x	root	root		130.1MB	31MAY2024:21:15:33
load_bigOrders.sas	-rwxrwxr-x	root	root		0.4KB	31MAY2024:21:15:21
employees.sas7bdat	-rwxrwxr-x	root	root		256.0KB	31MAY2024:21:15:21
create_bigOrders.sas	-rwxrwxr-x	root	root		1.2KB	31MAY2024:21:15:20
orders_new.sas7bdat	-rwxrwxr-x	root	root		17.5MB	31MAY2024:21:15:22
country_lookup.sas7bdat	-rwxrwxr-x	root	root		72.0KB	31MAY2024:21:15:20
orders.sas7bdat	-rwxrwxr-x	root	root		1.6GB	31MAY2024:21:17:38

The caslib attributes and the data source files in the caslib are listed.

To see in memory tables in the caslib, run the following:

```
PROC CASUTIL INCASLIB="viyaCas";
     LIST tables;
QUIT;
```

Tables have not been loaded into memory, so only the caslib attributes are visible. The log contains the following message:

```
NOTE: No tables are available in caslib VIYACAS of Cloud Analytic Services.
```

LOADING DATA INTO MEMORY

In a traditional SAS 9 environment, we work with data source files such as Excel, SAS, CSV, and JSON. You can still work with these files on the Compute Server as usual. However, to use data on the CAS Server, you must load the data source files to in-memory tables. This can be done using either a **server-side load** or a **client-side load**.

- **Server-Side Load**: Use when the file is already in a caslib. This is the most efficient method and is performed using PROC CASUTIL.
- **Client-Side Load**: Use when the file is on the Compute Server and needs to be transferred to CAS. This can be done using PROC CASUTIL, the DATA step, or PROC IMPORT.

Throughout this paper:

- The term **"file"** refers exclusively to a data source file.
- The term **"table"** or **"in-memory table"** refers exclusively to a data source file that has been loaded into memory.

These terms are **not** interchangeable.

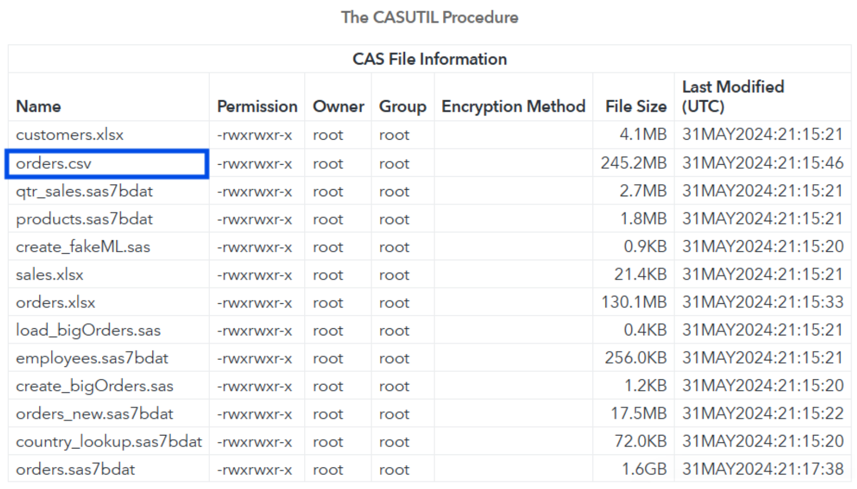
In-memory tables are temporary **copies** of data source files. Data source files remain unchanged on disk.

SERVER-SIDE LOAD

A **server-side load** is used when the data source file is already in a caslib. Since the file is accessible to the CAS Server, data can be loaded directly into memory for processing using the CASUTIL procedure. This method is recommended for loading large datasets and working with data across other SAS Viya applications.

Example: Server-Side Load Using PROC CASUTIL

Currently, **orders.csv** is stored in the **viyaCas** caslib, making this a server-side file.



The CASUTIL Procedure

CAS File Information

Name	Permission	Owner	Group	Encryption Method	File Size	Last Modified (UTC)
customers.xlsx	-rwxrwxr-x	root	root		4.1MB	31MAY2024:21:15:21
orders.csv	-rwxrwxr-x	root	root		245.2MB	31MAY2024:21:15:46
qtr_sales.sas7bdat	-rwxrwxr-x	root	root		2.7MB	31MAY2024:21:15:21
products.sas7bdat	-rwxrwxr-x	root	root		1.8MB	31MAY2024:21:15:21
create_fakeML.sas	-rwxrwxr-x	root	root		0.9KB	31MAY2024:21:15:20
sales.xlsx	-rwxrwxr-x	root	root		21.4KB	31MAY2024:21:15:21
orders.xlsx	-rwxrwxr-x	root	root		130.1MB	31MAY2024:21:15:33
load_bigOrders.sas	-rwxrwxr-x	root	root		0.4KB	31MAY2024:21:15:21
employees.sas7bdat	-rwxrwxr-x	root	root		256.0KB	31MAY2024:21:15:21
create_bigOrders.sas	-rwxrwxr-x	root	root		1.2KB	31MAY2024:21:15:20
orders_new.sas7bdat	-rwxrwxr-x	root	root		17.5MB	31MAY2024:21:15:22
country_lookup.sas7bdat	-rwxrwxr-x	root	root		72.0KB	31MAY2024:21:15:20
orders.sas7bdat	-rwxrwxr-x	root	root		1.6GB	31MAY2024:21:17:38

The following example loads **orders.csv** into memory as **ordersIM** (IM to distinguish in-memory) in the **viyaCas** caslib:

```
PROC CASUTIL;
   LOAD CASDATA="orders.csv"
   INCASLIB="viyacas"
   CASOUT="ordersIM"
   OUTCASLIB="viyaCas";
QUIT;
```

- **CASDATA=** defines the data source file to load into memory.
- **INCASLIB=** defines the caslib the data source file is currently in.
- **CASOUT=** defines the name of the new in-memory table.
- **OUTCASLIB=** defines the caslib the new in-memory table will be in. This caslib does not have to match the caslib the data source file is currently in.

```
NOTE: Cloud Analytic Services made the file orders.csv available as table ORDERSIM in caslib viyaCas.
```

To view the table in the **viyacas** caslib, use the LIST TABLES statement in PROC CASUTIL:

```
PROC CASUTIL INCASLIB="viyaCas";
     LIST tables;
QUIT;
```

Partial Results:

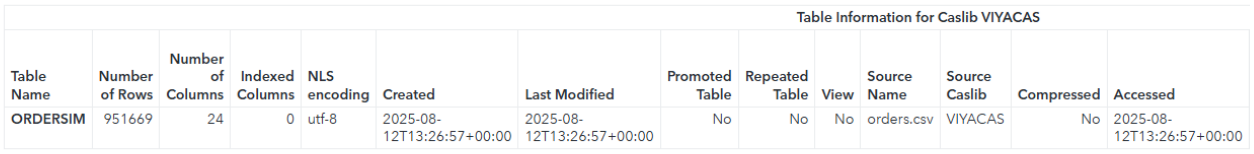


Table Information for Caslib VIYACAS													
Table Name	Number of Rows	Number of Columns	Indexed Columns	NLS encoding	Created	Last Modified	Promoted Table	Repeated Table	View	Source Name	Source Caslib	Compressed	Accessed
ORDERSIM	951669	24	0	utf-8	2025-08-12T13:26:57+00:00	2025-08-12T13:26:57+00:00	No	No	No	orders.csv	VIYACAS	No	2025-08-12T13:26:57+00:00

By default, in-memory tables are **session scope**, meaning they are available only within the current session and are dropped when the CAS session ends. Right now, **ordersIM** is only visible in SAS Studio. To make a table available across SAS Viya applications, use the PROMOTE option, making it **global scope**:

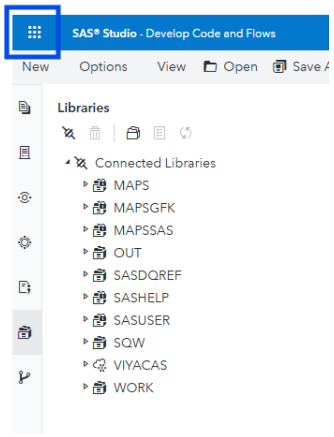
```
LIBNAME casuser CAS CASLIB=casuser;
PROC CASUTIL;
   LOAD CASDATA="orders.csv"
   INCASLIB="viyacas"
   CASOUT="ordersIM_global"
   OUTCASLIB="casuser"
   PROMOTE;
QUIT;
```

In SAS Viya for Learners (VFL), users do not have permission to create global caslibs. To promote a table to global scope, the caslib it is promoted to bust also be global. In VFL, the **casuser** caslib is a predefined global caslib. I used this to create the global scope table, **ordersIM_global**. I also mapped the **casuser** caslib to the **casuser** libref to view in the Libraries pane and future examples.

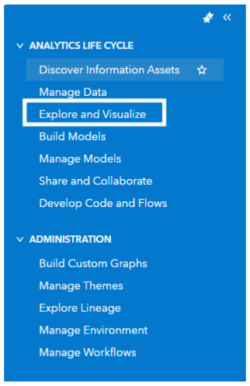
Now, the table is available in other applications. We can verify this in **Visual Analytics**.

Navigate to:

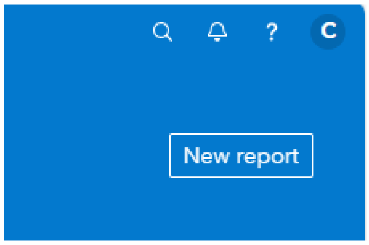
Applications menu



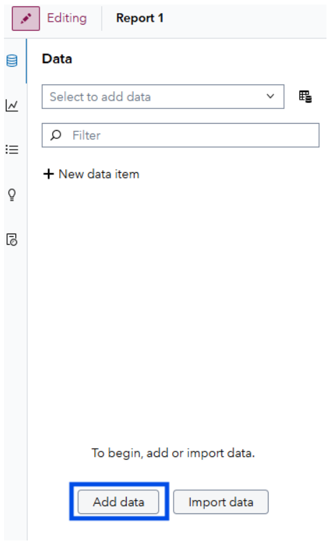
Explore and Visualize



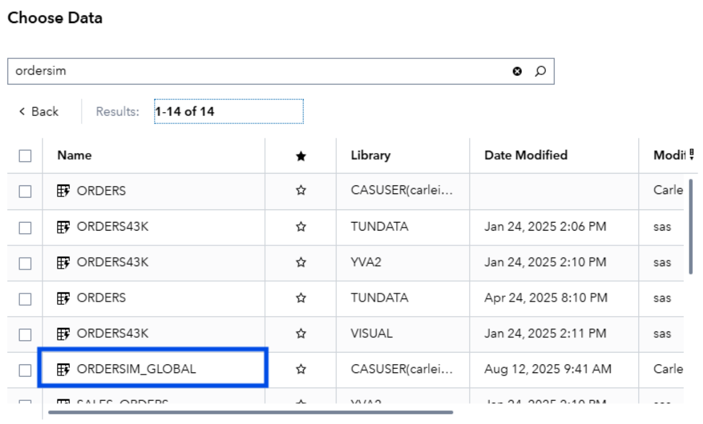
New Report



Add Data



Search for **ordersIM_global**. The table will be visible and available to use for reporting.


Choose Data

ordersim

Back Results: 1-14 of 14

Name	★	Library	Date Modified	Modi
ORDERS	☆	CASUSER(carlei…		Carle
ORDERS43K	☆	TUNDATA	Jan 24, 2025 2:06 PM	sas
ORDERS43K	☆	YVA2	Jan 24, 2025 2:10 PM	sas
ORDERS	☆	TUNDATA	Apr 24, 2025 8:10 PM	sas
ORDERS43K	☆	VISUAL	Jan 24, 2025 2:11 PM	sas
ORDERSIM_GLOBAL	☆	CASUSER(carlei…	Aug 12, 2025 9:41 AM	Carle

CLIENT-SIDE LOAD

A **client-side load** is used when the data source file is on the Compute Server (e.g., in a SAS library or folder path) and needs to be transferred to the CAS Server. This method is great if you are a traditional SAS programmer and want to continue using traditional SAS code to prepare data, then load the data into memory. Once the data is loaded into memory, you can access it in other Viya applications like Visual Analytics for reporting. This method is more resource-intensive, so a server-side load is preferred whenever possible.

There are three ways to perform a client-side load:

1. PROC CASUTIL
2. DATA step
3. PROC IMPORT

Example: Client-Side Load Using PROC CASUTIL- SAS Data Set

This example loads the **sashelp.cars** dataset into memory as **cars_CAS** in the **casuser** caslib:

```
PROC CASUTIL;
   LOAD DATA=sashelp.cars
   CASOUT="cars_CAS"
   OUTCASLIB="casuser"
   PROMOTE;
QUIT;
```

- **DATA=** defines the library and table name to load into memory.
- **CASOUT=** defines the name of the new in-memory table.
- **OUTCASLIB=** defines the caslib the new in-memory table will be in.
- **PROMOTE** ensures the table is global scope.

To view the table in **casuser** run:

```
PROC CASUTIL INCASLIB="casuser";
     LIST tables;
QUIT;
```

Partial Results:

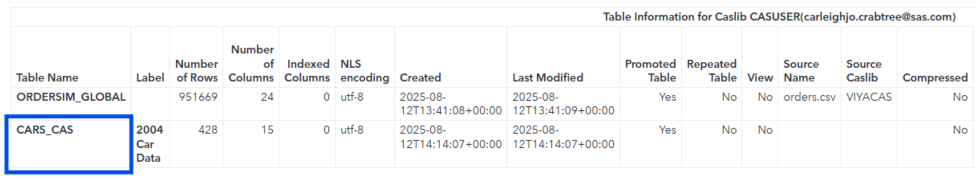
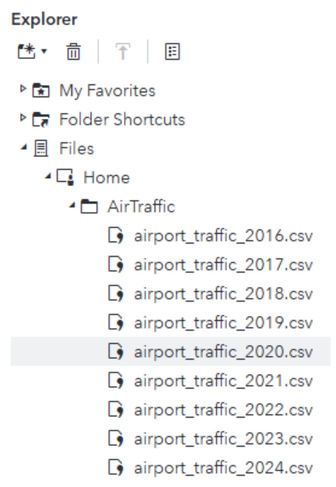


Table Information for Caslib CASUSER(carleighjo.crabtree@sas.com)

Table Name	Label	Number of Rows	Number of Columns	Indexed Columns	NLS encoding	Created	Last Modified	Promoted Table	Repeated Table	View	Source Name	Source Caslib	Compressed
ORDERSIM_GLOBAL		951669	24	0	utf-8	2025-08-12T13:41:08+00:00	2025-08-12T13:41:09+00:00	Yes	No	No	orders.csv	VIYACAS	No
CARS_CAS	2004 Car Data	428	15	0	utf-8	2025-08-12T14:14:07+00:00	2025-08-12T14:14:07+00:00	Yes	No	No			No

Example: Client-Side Load Using PROC CASUTIL- CSV File

Airport_traffic_2020.csv is currently a client-side file located in the **AirTraffic** folder.



Explorer

- My Favorites
- Folder Shortcuts
- Files
  - Home
    - AirTraffic
      - airport_traffic_2016.csv
      - airport_traffic_2017.csv
      - airport_traffic_2018.csv
      - airport_traffic_2019.csv
      - airport_traffic_2020.csv
      - airport_traffic_2021.csv
      - airport_traffic_2022.csv
      - airport_traffic_2023.csv
      - airport_traffic_2024.csv

This example loads **airport_traffic_2020.csv** into memory and creates **EU_Air_2020** in the **casuser** caslib:

```
PROC CASUTIL;

   LOAD
FILE="/export/viya/homes/carleighjo.crabtree@sas.com/AirTraffic/airport_traff
ic_2020.csv"

   CASOUT="EU_Air_2020"

   OUTCASLIB="casuser"

   PROMOTE;

QUIT;
```

- **FILE=** defines the path, file name and file extension of the file to load into memory.

Partial LIST tables Results:

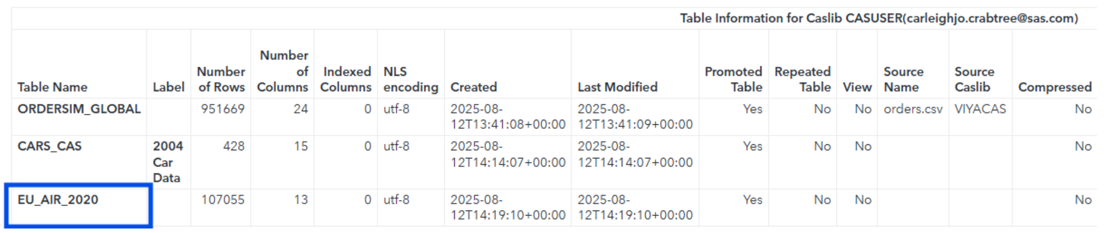


Table Information for Caslib CASUSER(carleighjo.crabtree@sas.com)

Table Name	Label	Number of Rows	Number of Columns	Indexed Columns	NLS encoding	Created	Last Modified	Promoted Table	Repeated Table	View	Source Name	Source Caslib	Compressed
ORDERSIM_GLOBAL		951669	24	0	utf-8	2025-08-12T13:41:08+00:00	2025-08-12T13:41:09+00:00	Yes	No	No	orders.csv	VIYACAS	No
CARS_CAS	2004 Car Data	428	15	0	utf-8	2025-08-12T14:14:07+00:00	2025-08-12T14:14:07+00:00	Yes	No	No			No
EU_AIR_2020		107055	13	0	utf-8	2025-08-12T14:19:10+00:00	2025-08-12T14:19:10+00:00	Yes	No	No			No

Example: Client-Side Load Using the DATA Step

Data can also be loaded into memory using the DATA step by specifying a libref mapped to a caslib for the output data set. The following example filters the SASHELP.CARS table for cars made by Toyota and writes the results to the in-memory table **ToyotaCars_CAS** in the **casuser** caslib.

```
DATA casuser.ToyotaCars_CAS (PROMOTE=YES);
   SET sashelp.cars;
   WHERE Make="Toyota";
RUN;
```

- Defining the **casuser** caslib ensures that **ToyotaCars_CAS** is an in-memory table.
- **(PROMOTE=YES)** ensures the table is global scope.

Partial LIST tables Results:

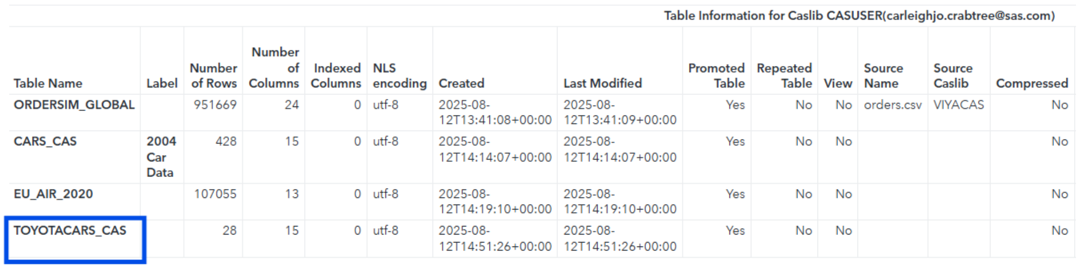


Table Information for Caslib CASUSER(carleighjo.crabtree@sas.com)

Table Name	Label	Number of Rows	Number of Columns	Indexed Columns	NLS encoding	Created	Last Modified	Promoted Table	Repeated Table	View	Source Name	Source Caslib	Compressed
ORDERSIM_GLOBAL		951669	24	0	utf-8	2025-08-12T13:41:08+00:00	2025-08-12T13:41:09+00:00	Yes	No	No	orders.csv	VIYACAS	No
CARS_CAS	2004 Car Data	428	15	0	utf-8	2025-08-12T14:14:07+00:00	2025-08-12T14:14:07+00:00	Yes	No	No			No
EU_AIR_2020		107055	13	0	utf-8	2025-08-12T14:19:10+00:00	2025-08-12T14:19:10+00:00	Yes	No	No			No
TOYOTACARS_CAS		28	15	0	utf-8	2025-08-12T14:51:26+00:00	2025-08-12T14:51:26+00:00	Yes	No	No			No

Example: Client-Side Load Using PROC IMPORT

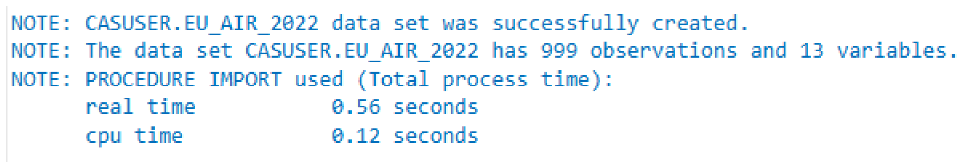
PROC IMPORT can be used to load files like CSVs, TXT or Excel. In the OUT=option, specify a libref that is mapped to a caslib as the library for the output table.

The following example loads **airport_traffic_2022.csv** into memory as **EU_Air_2022** in the **casuser** caslib.

```
OPTIONS OBS=1000;


PROC IMPORT DATAFILE="
/export/viya/homes/carleighjo.crabtree@sas.com/AirTraffic/airport_traffic_202
2.csv"
           DBMS=csv
           OUT=casuser.EU_Air_2022
           REPLACE;
RUN;


OPTIONS OBS=max;
```

- **OPTIONS OBS=** limits the number of observations that are imported.
- **DATAFILE=** defines the path, file name and file extension of the file to load into memory.
- **OUT=** defines the caslib and in-memory table name.

Partial Log:



```
NOTE: CASUSER.EU_AIR_2022 data set was successfully created.
NOTE: The data set CASUSER.EU_AIR_2022 has 999 observations and 13 variables.
NOTE: PROCEDURE IMPORT used (Total process time):
      real time           0.56 seconds
      cpu time            0.12 seconds
```

Partial LIST tables Results:

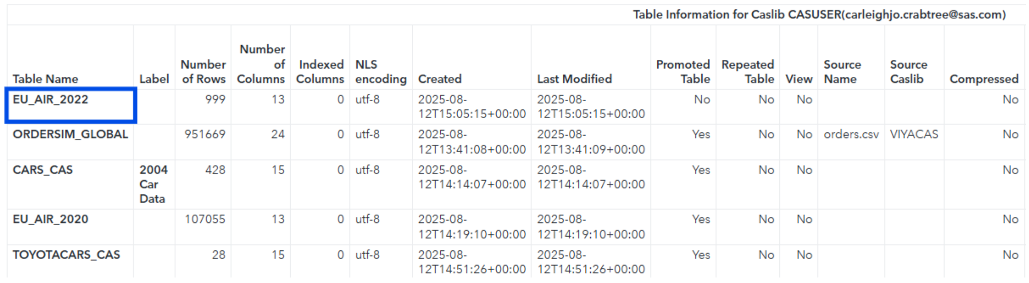


Table Information for Caslib CASUSER(carleighjo.crabtree@sas.com)

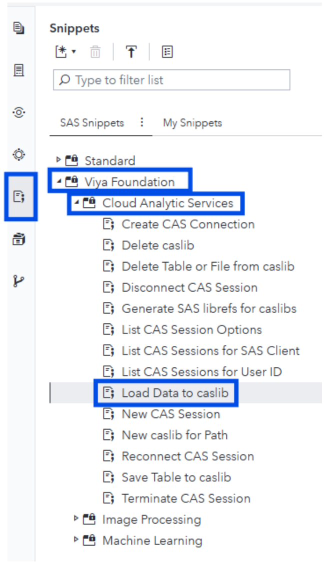
Table Name	Label	Number of Rows	Number of Columns	Indexed Columns	NLS encoding	Created	Last Modified	Promoted Table	Repeated Table	View	Source Name	Source Caslib	Compressed
EU_AIR_2022		999	13	0	utf-8	2025-08-12T15:05:15+00:00	2025-08-12T15:05:15+00:00	No	No	No			No
ORDERSIM_GLOBAL		951669	24	0	utf-8	2025-08-12T13:41:08+00:00	2025-08-12T13:41:09+00:00	Yes	No	No	orders.csv	VIYACAS	No
CARS_CAS	2004 Car Data	428	15	0	utf-8	2025-08-12T14:14:07+00:00	2025-08-12T14:14:07+00:00	Yes	No	No			No
EU_AIR_2020		107055	13	0	utf-8	2025-08-12T14:19:10+00:00	2025-08-12T14:19:10+00:00	Yes	No	No			No
TOYOTACARS_CAS		28	15	0	utf-8	2025-08-12T14:51:26+00:00	2025-08-12T14:51:26+00:00	Yes	No	No			No

Load Data to Caslib Snippet

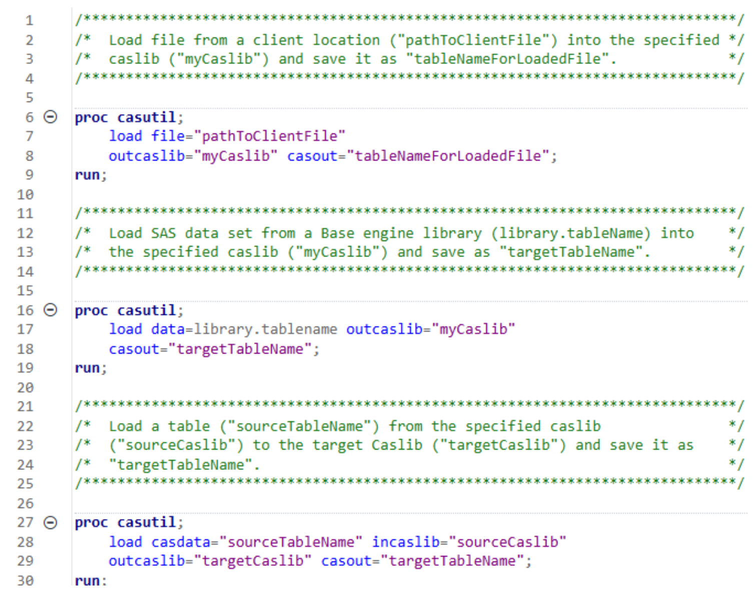
Whether you're performing a server-side load or a client-side load, PROC CASUTIL is the most efficient option for loading files to in-memory tables. In SAS Studio, there is a snippet with PROC CASUTIL syntax for loading a client-side file, SAS table in a library, and server-side file into memory.

To use this snippet, navigate to:

Snippets> Viya Foundation> Cloud Analytic Services> Load Data to caslib



Modify the code in the snippet depending on the use-case.


```
/*****************************************************************************/
/*  Load file from a client location ("pathToClientFile") into the specified */
/*  caslib ("myCaslib") and save it as "tableNameForLoadedFile".             */
/*****************************************************************************/

proc casutil;
    load file="pathToClientFile"
    outcaslib="myCaslib" casout="tableNameForLoadedFile";
run;

/*****************************************************************************/
/*  Load SAS data set from a Base engine library (library.tableName) into    */
/*  the specified caslib ("myCaslib") and save as "targetTableName".         */
/*****************************************************************************/

proc casutil;
    load data=library.tablename outcaslib="myCaslib"
    casout="targetTableName";
run;

/*****************************************************************************/
/*  Load a table ("sourceTableName") from the specified caslib               */
/*  ("sourceCaslib") to the target Caslib ("targetCaslib") and save it as    */
/*  "targetTableName".                                                       */
/*****************************************************************************/

proc casutil;
    load casdata="sourceTableName" incaslib="sourceCaslib"
    outcaslib="targetCaslib" casout="targetTableName";
run;
```

ALTERING THE DATA STEP TO RUN IN CAS

Once data has been loaded into memory, it becomes available for use on the CAS server. This means in-memory data in a caslib can be accessed using the DATA step, procedures, or in other SAS Viya applications. It's important to understand how the DATA step processes data on the CAS server, helpful options, and considerations when using the SUM statement or grouping data with BY variables.

CONSIDERATIONS AND HELPFUL OPTIONS

Before running a DATA step on the CAS server:

1. Start a CAS session.
2. Ensure there are in-memory tables in a caslib connected to a libref.

After this has been done, both the input and output tables must specify a caslib and the DATA step must be fully **CAS enabled**. Some restrictions can cause the DATA step to run on Compute instead of CAS. The SESSREF= option on the DATA statement can be used to specify to only run the DATA step in CAS. If it is not CAS enabled, an error will be generated.

Before the examples to come, the **chocolate_enterprise_reporting.sas7bdat** file has been loaded into memory in the **casuser** caslib and a library named CHOC has been created:

```
libname choc "/export/viya/homes/carleighjo.crabtree@sas.com/Choc";

PROC CASUTIL;

      /*data source file name*/

      LOAD CASDATA="chocolate_enterprise_reporting.sas7bdat"

      /*caslib the data source file is currently in*/

      INCASLIB="casuser"

      /*name of the new in-memory table*/

      CASOUT="Choc_Enterprise"

      /*caslib to put the in-memory table in*/

      /*can be the same or different from the INCASLIB*/

      OUTCASLIB="casuser"
```

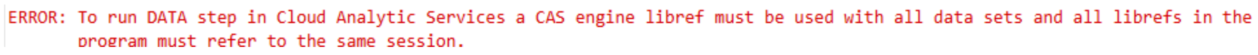
```
     promote;
QUIT;
```

Example: Using SESSREF and MSGLEVEL Options

The following DATA step attempts to create **EU_Orders** in the **work** library, rather than a caslib. With the SESSREF= option, the step fails and produces an error.

```
data work.EU_Orders /sessref= CJsSession;
     set casuser.choc_enterprise;
     where continent="Europe";
run;
```

Partial Log:

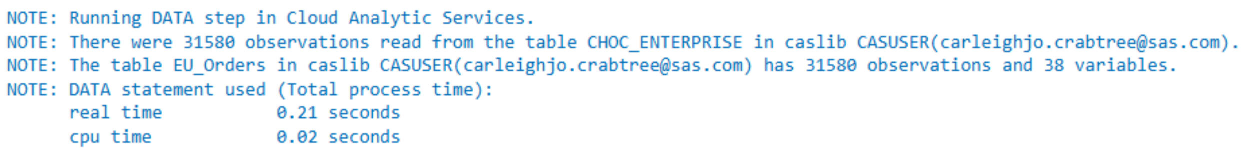


```
ERROR: To run DATA step in Cloud Analytic Services a CAS engine libref must be used with all data sets and all librefs in the
       program must refer to the same session.
```

Once the **work** library is changed to the caslib **casuser**, the DATA step runs in CAS as expected.

```
data casuser.EU_Orders /sessref=CJsSession;
     set casuser.choc_enterprise;
     where continent="Europe";
run;
```

Partial Log:

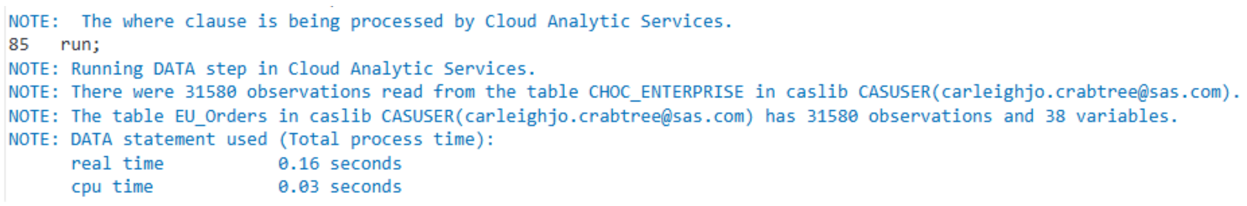


```
NOTE: Running DATA step in Cloud Analytic Services.
NOTE: There were 31580 observations read from the table CHOC_ENTERPRISE in caslib CASUSER(carleighjo.crabtree@sas.com).
NOTE: The table EU_Orders in caslib CASUSER(carleighjo.crabtree@sas.com) has 31580 observations and 38 variables.
NOTE: DATA statement used (Total process time):
      real time           0.21 seconds
      cpu time            0.02 seconds
```

Another useful system option to generate in-depth messages in the log is MSGLVL=i. When added before the same code as the previous example, the log contains an additional note confirming the WHERE clause ran in CAS.

```
options msglevel=i;


data casuser.EU_Orders /sessref=CJsSession;
     set casuser.choc_enterprise;
     where continent="Europe";
run;
```

Partial Log:



```
NOTE:  The where clause is being processed by Cloud Analytic Services.
85   run;
NOTE: Running DATA step in Cloud Analytic Services.
NOTE: There were 31580 observations read from the table CHOC_ENTERPRISE in caslib CASUSER(carleighjo.crabtree@sas.com).
NOTE: The table EU_Orders in caslib CASUSER(carleighjo.crabtree@sas.com) has 31580 observations and 38 variables.
NOTE: DATA statement used (Total process time):
      real time           0.16 seconds
      cpu time            0.03 seconds
```

BEHIND THE SCENES

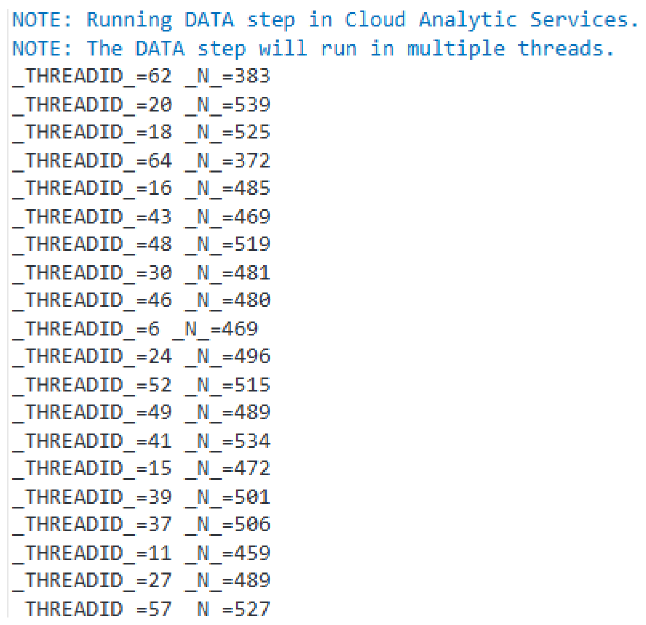
When the DATA step runs in CAS, data is distributed among multiple threads. CAS automatically determines how many rows of data go to each thread, and the DATA step runs independently on each.

The number of threads varies by environment. Use PUT _THREADID_= _N_ to visualize how the data has been distributed.

Example: Visualizing Threads

```
data casuser.EU_Orders;

     set casuser.choc_enterprise end=eof;

     where continent="Europe";

     if eof then put _threadid_= _N_=;

run;
```

Partial Log:



```
NOTE: Running DATA step in Cloud Analytic Services.
NOTE: The DATA step will run in multiple threads.
_THREADID_=62 _N_=383
_THREADID_=20 _N_=539
_THREADID_=18 _N_=525
_THREADID_=64 _N_=372
_THREADID_=16 _N_=485
_THREADID_=43 _N_=469
_THREADID_=48 _N_=519
_THREADID_=30 _N_=481
_THREADID_=46 _N_=480
_THREADID_=6 _N_=469
_THREADID_=24 _N_=496
_THREADID_=52 _N_=515
_THREADID_=49 _N_=489
_THREADID_=41 _N_=534
_THREADID_=15 _N_=472
_THREADID_=39 _N_=501
_THREADID_=37 _N_=506
_THREADID_=11 _N_=459
_THREADID_=27 _N_=489
 THREADID =57  N =527
```

The rows in CASUSER.CHOC_ENTERPRISE were distributed to multiple threads- about 500 rows per thread. Each thread completes processing at different times, which is why the threads are not in sequential order in the log. As threads complete processing, they return to their original position on a linked list to maintain row order. This distribution of data is known as **multithreaded processing**.

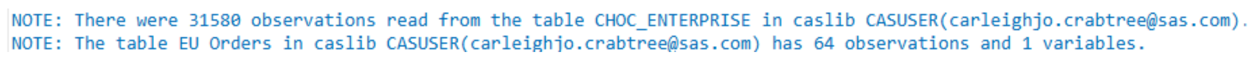
SUM STATEMENT

Distributing data works great for speed, but how might this effect a program with an accumulating column? The sum statement is considered a **full table operation**, meaning the entire table must be analyzed to produce an accurate result.

In the program below, we would typically expect one row with the total number of orders placed in France. However, this program produces 64 rows. This is because the data was distributed among 64 threads and each thread calculated the sum independently. Results from each thread were returned to the controller, compiled and output to the CASUSER.EU_ORDERS data set.

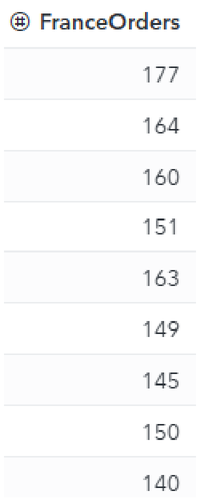
```
data casuser.EU_Orders;

     set casuser.choc_enterprise end=eof;

     where continent="Europe";

     if country_nm="France" then FranceOrders+1;

     if eof=1 then output;
     keep FranceOrders;
     if eof then put _threadid_= _N_=;
run;
```

Partial Log:



```
NOTE: There were 31580 observations read from the table CHOC_ENTERPRISE in caslib CASUSER(carleighjo.crabtree@sas.com).
NOTE: The table EU Orders in caslib CASUSER(carleighjo.crabtree@sas.com) has 64 observations and 1 variables.
```

Partial Output Data:



FranceOrders
177
164
160
151
163
149
145
150
140

There are a few ways to remedy these unexpected results:

1. Run a *second*, single-threaded DATA step
2. Run the *original* DATA step single-threaded
3. Use the Compute Server

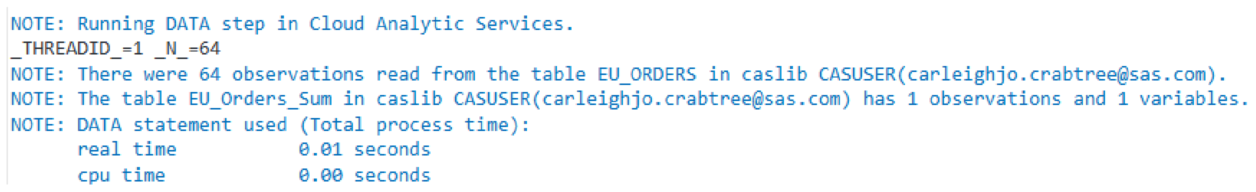
Option 1: Run a Second, Single-Threaded DATA Step

On the CAS server, the DATA step runs multithreaded by default, but it can also run single threaded. To run the DATA step single threaded, add SINGLE=YES to the DATA statement. When working with large data, the recommended approach is to write the first DATA step multithreaded, and the second single threaded to summarize the condensed results.

The following DATA step uses the summarized EU_ORDERS table created above and creates EU_ORDERS_SUM to generate the final total of France orders. Because the data was already summarized to 64 rows, running a second step single threaded is very efficient (0.01 seconds).

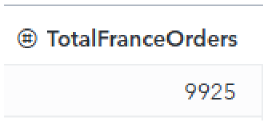
```
data casuser.EU_Orders_Sum / single=yes;
     set casuser.eu_orders end=eof;
     TotalFranceOrders+FranceOrders;
     keep TotalFranceOrders;
     if eof=1 then output;
     if eof then put _threadid_= _N_=;
run;
```

Partial Log:



```
NOTE: Running DATA step in Cloud Analytic Services.
_THREADID_=1 _N_=64
NOTE: There were 64 observations read from the table EU_ORDERS in caslib CASUSER(carleighjo.crabtree@sas.com).
NOTE: The table EU_Orders_Sum in caslib CASUSER(carleighjo.crabtree@sas.com) has 1 observations and 1 variables.
NOTE: DATA statement used (Total process time):
      real time           0.01 seconds
      cpu time            0.00 seconds
```

Output Data:



TotalFranceOrders
9925

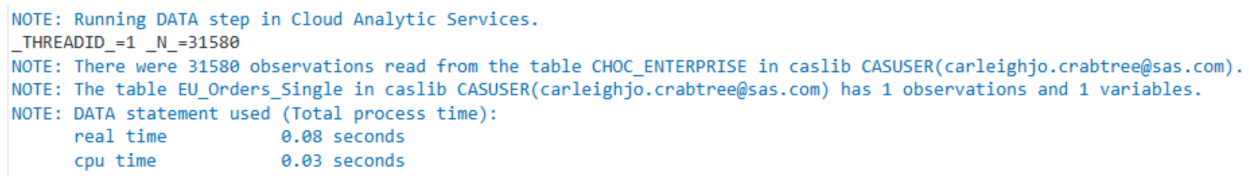
Option 2: Run the Original DATA Step Single-Threaded

In the original DATA step, add / SINGLE=YES to the DATA statement. This avoids writing a second DATA step, but may increase run time.

The following DATA step uses the original CHOC_ENTERPRISE table and creates EU_ORDERS with one row totaling the number of France orders in one step. Notice the step took longer than the previous method (0.08 seconds). While this may not be a significant time savings on this data size, it can vary with larger tables.

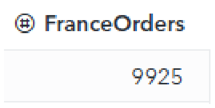
```
data casuser.EU_Orders_Single / single=yes;
      set casuser.choc_enterprise end=eof;
      where continent="Europe";
      if country_nm="France" then FranceOrders+1;
      if eof=1 then output;
      keep FranceOrders;
      if eof then put _threadid_= _N_=;
run;
```

Partial Log:



```
NOTE: Running DATA step in Cloud Analytic Services.
_THREADID_=1 _N_=31580
NOTE: There were 31580 observations read from the table CHOC_ENTERPRISE in caslib CASUSER(carleighjo.crabtree@sas.com).
NOTE: The table EU_Orders_Single in caslib CASUSER(carleighjo.crabtree@sas.com) has 1 observations and 1 variables.
NOTE: DATA statement used (Total process time):
      real time           0.08 seconds
      cpu time            0.03 seconds
```

Output Data:



FranceOrders
9925

Option 3: Use the Compute Server

Remember the DATA step can be run on the Compute Server without modifying code.

GROUPING DATA ON THE CAS SERVER

When grouping data on the CAS server, there is no need to presort. Data is distributed to multiple threads, however, each BY group remains together and is processed on the same thread. When using multiple variables on the BY statement, the first variable determines how rows are distributed to threads.

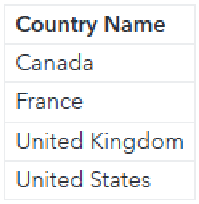
If the first BY variable has a format, CAS uses the formatted value for grouping. Also, note that the descending option is not supported for the first BY variable when running in CAS.

Example: Grouping Data with One BY Variable

The following PROC SQL step confirms there are four distinct values in the **country_nm** column:

```
proc sql;

select distinct country_nm

     from casuser.choc_enterprise;

quit;
```

Results:

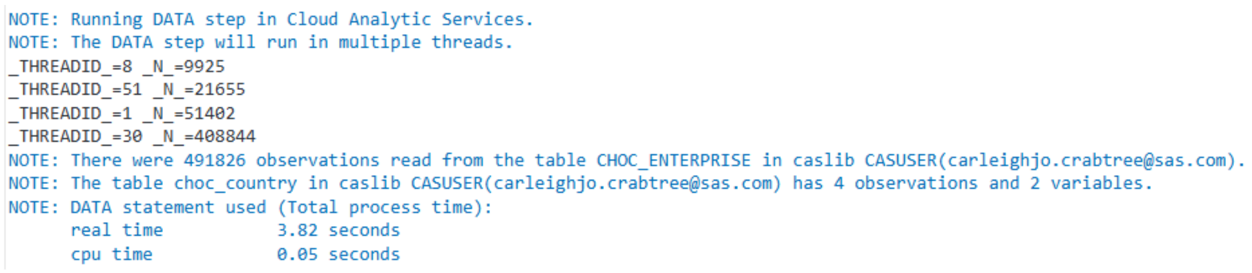


Country Name
Canada
France
United Kingdom
United States

The following DATA step contains a BY statement. Only four threads are used, one for each BY group value in **country_nm**. The DATA step produces four rows with the total number of orders placed in each country.

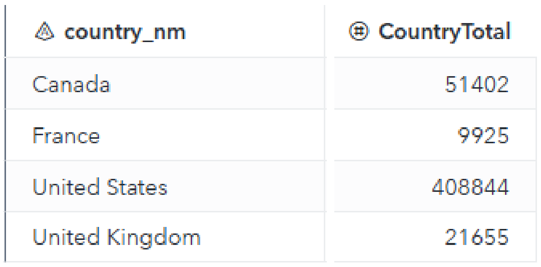
```
data casuser.choc_country;

     set casuser.choc_enterprise end=eof;

     by country_nm;

     if first.country_nm then CountryTotal=0;

     CountryTotal+1;

     if last.country_nm then output;

     keep country_nm CountryTotal;

     if eof then put _threadid_= _N_=;

run;
```

Partial Log:



```
NOTE: Running DATA step in Cloud Analytic Services.
NOTE: The DATA step will run in multiple threads.
_THREADID_=8 _N_=9925
_THREADID_=51 _N_=21655
_THREADID_=1 _N_=51402
_THREADID_=30 _N_=408844
NOTE: There were 491826 observations read from the table CHOC_ENTERPRISE in caslib CASUSER(carleighjo.crabtree@sas.com).
NOTE: The table choc_country in caslib CASUSER(carleighjo.crabtree@sas.com) has 4 observations and 2 variables.
NOTE: DATA statement used (Total process time):
      real time           3.82 seconds
      cpu time            0.05 seconds
```

Output Data:



country_nm	CountryTotal
Canada	51402
France	9925
United States	408844
United Kingdom	21655

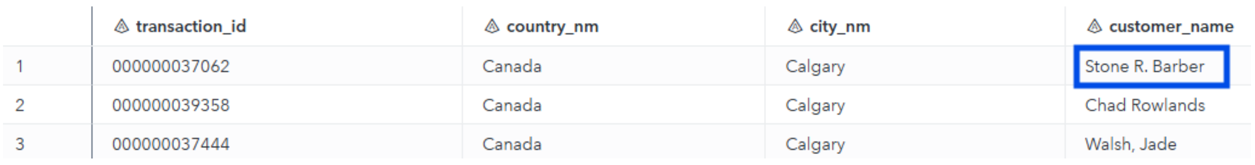
Example: Grouping Data with Multiple BY Variables

While all values within the *first* BY group stay together on one thread, the row order within a group can vary. This may lead to different results each time the DATA step runs.

The following DATA step is grouped by two variables: **country_nm** and **city_nm**. The first time the step is run, the first row contains the customer *Stone R. Barber*. On the second run, the first row contains the customer *Edwards, Claire*.

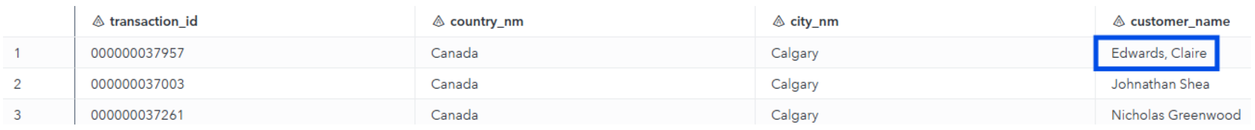
```
data casuser.choc_city;
     set casuser.choc_enterprise end=eof;
     by country_nm city_nm;
     keep country_nm city_nm transaction_id customer_name;
     if eof then put _threadid_= _N_=;
run;
```

Run 1 Partial Output Data:



	transaction_id	country_nm	city_nm	customer_name
1	000000037062	Canada	Calgary	Stone R. Barber
2	000000039358	Canada	Calgary	Chad Rowlands
3	000000037444	Canada	Calgary	Walsh, Jade

Run 2 Partial Output Data:



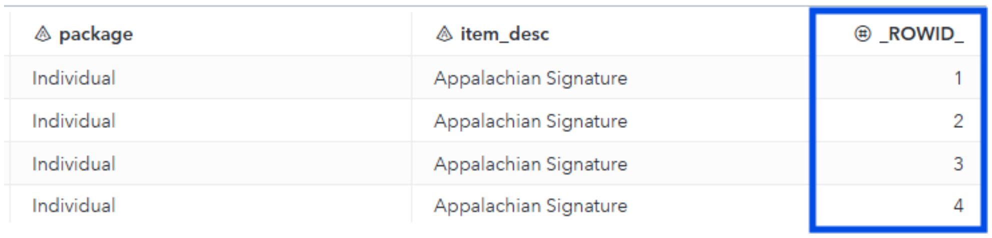
	transaction_id	country_nm	city_nm	customer_name
1	000000037957	Canada	Calgary	Edwards, Claire
2	000000037003	Canada	Calgary	Johnathan Shea
3	000000037261	Canada	Calgary	Nicholas Greenwood

To maintain row order, use the ADDROWID option when loading data into CAS. This adds a column to the output data set named **_ROWID_** that increments in numeric value with each row. These values will remain unchanged.

The following DATA step reads the table **chocolate_enterprise_reporting** in the **choc** library and creates the **choc_rowid** in-memory table in the **casuser** caslib. Because of the ADDROWID option, **_ROWID_** is added as the last column.

```
data casuser.choc_rowid (addRowId=yes);
     set choc.chocolate_enterprise_reporting;
run;
```

Partial Output Data:



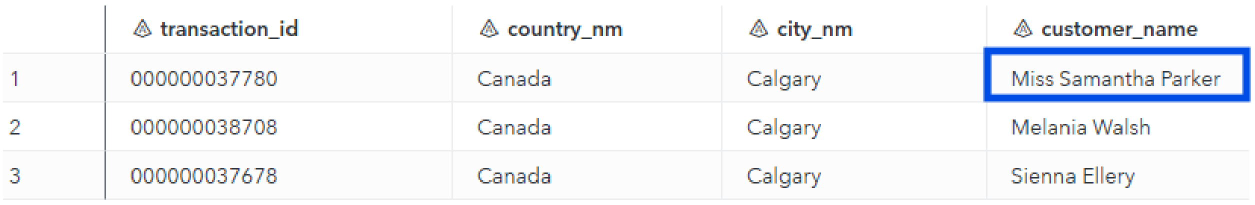
package	item_desc	_ROWID_
Individual	Appalachian Signature	1
Individual	Appalachian Signature	2
Individual	Appalachian Signature	3
Individual	Appalachian Signature	4

When grouping **casuser.choc_rowid** with multiple variables, **_ROWID_** can be used to ensure the rows are read in order within the groups.

In the following DATA step, **_ROWID_** is not used on the BY statement. When run multiple times, the order of the rows within the groups of **country_nm** and **city_nm** is not guaranteed. This can easily be seen as *Miss Samantha Parker* is the first row on the first run, but *Leslie Swan* is the first row on the second run of the same DATA step.

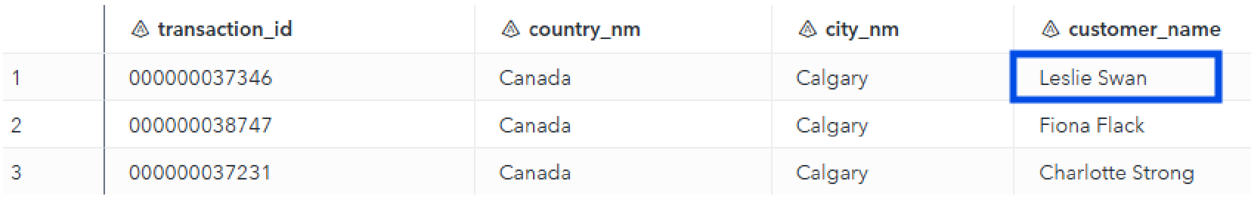
```
data casuser.withoutrowid;
     set casuser.choc_rowid end=eof;
     by country_nm city_nm;
     keep country_nm city_nm transaction_id customer_name;
     if eof then put _threadid_= _N_=;
run;
```

Run 1 Partial Output Data:



	transaction_id	country_nm	city_nm	customer_name
1	000000037780	Canada	Calgary	Miss Samantha Parker
2	000000038708	Canada	Calgary	Melania Walsh
3	000000037678	Canada	Calgary	Sienna Ellery

Run 2 Partial Output Data:

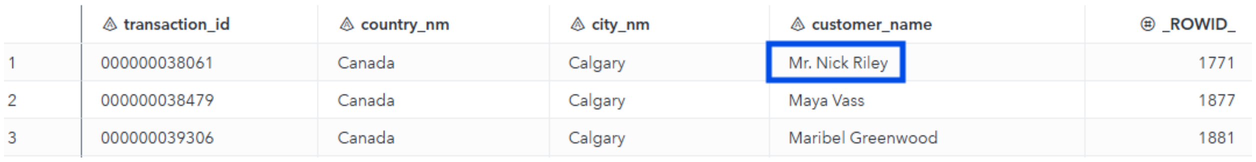


	transaction_id	country_nm	city_nm	customer_name
1	000000037346	Canada	Calgary	Leslie Swan
2	000000038747	Canada	Calgary	Fiona Flack
3	000000037231	Canada	Calgary	Charlotte Strong

When **_rowid_** is added to the DATA step as the final variable on the BY statement, the original data set row order within the BY groups is maintained. This can be seen as *Mr. Nick Riley* is the first customer after the first and second run of the DATA step.

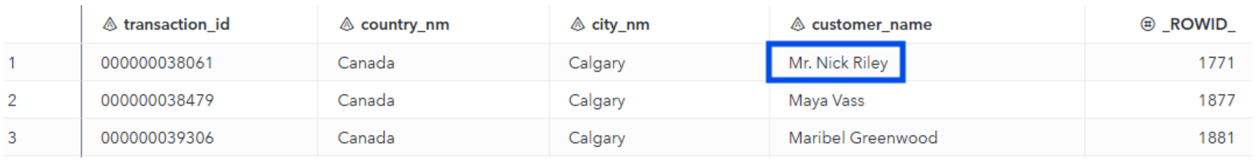
```
data casuser.withrowid;
     set casuser.choc_rowid end=eof;
     by country_nm city_nm _rowid_;
     keep country_nm city_nm transaction_id customer_name _rowid_;
     if eof then put _threadid_= _N_=;
run;
```

Run 1 Partial Output Data:



	transaction_id	country_nm	city_nm	customer_name	_ROWID_
1	000000038061	Canada	Calgary	Mr. Nick Riley	1771
2	000000038479	Canada	Calgary	Maya Vass	1877
3	000000039306	Canada	Calgary	Maribel Greenwood	1881

Run 2 Partial Output Data:



	transaction_id	country_nm	city_nm	customer_name	_ROWID_
1	000000038061	Canada	Calgary	Mr. Nick Riley	1771
2	000000038479	Canada	Calgary	Maya Vass	1877
3	000000039306	Canada	Calgary	Maribel Greenwood	1881

CONCLUSION

In summary, SAS Viya allows you to submit traditional SAS code on the Compute Server, just as you always have. It also introduces a more powerful, flexible architecture than traditional SAS 9 through its CAS Server and in-memory, massively parallel processing capabilities. While the Compute Server offers compatibility with your existing SAS programs, the CAS Server is the game-changer for handling larger data and more demanding computations.

REFERENCES

Kubernetes Documentation:

Overview | Kubernetes

SAS Documentation:

SAS Help Center: Introduction to the SAS Viya Platform

SAS Help Center: SAS Viya Platform Fundamentals

SAS Help Center: Caslibs

SAS Help Center: Accessing Data with SAS Cloud Analytic Services

SAS Help Center: CAS Statement

SAS Help Center: CASLIB Statement

SAS Help Center: LIBNAME Statement: CAS Engine

SAS Help Center: CAS LIBNAME Engine

SAS Help Center: CASUTIL Procedure

SAS Help Center: Syntax: PROC CASUTIL LIST Statement

SAS Help Center: Syntax: PROC CASUTIL LOAD Statement

SAS Help Center: Syntax: PROC CASUTIL PROMOTE Statement

SAS Help Center: Restrictions and Supported Language Elements

SAS Help Center: DATA Statement

SAS Help Center: DATA Step Processing in CAS

SAS Help Center: DATA Statement

SAS Help Center: Grouping and Ordering

SAS Help Center: ADDROWID= Data Set Option

CONTACT INFORMATION

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

Carleigh Jo Crabtree
CarleighJo.Crabtree@sas.com
www.linkedin.com/in/carleigh-jo-crabtree