THE POWER TO KNOW®

Programming for SAS® Viya®

Course Notes

Programming for SAS® Viya® Course Notes was developed by Davetta Dunlap and Stacey Syphus. Additional contributions were made by Brittany Coleman, Brian Gayle, Mark Jordan, Jim Simon, and Anna Yarbrough. Instructional design, editing, and production support was provided by the Learning Design and Development team.

Programming for SAS® Viya® Course Notes

Book code E71705, course code LWPGVY35/PGVY35, prepared date 16Jul2020. LWPGVY35_001

ISBN 978-1-951686-11-6

Table of Contents

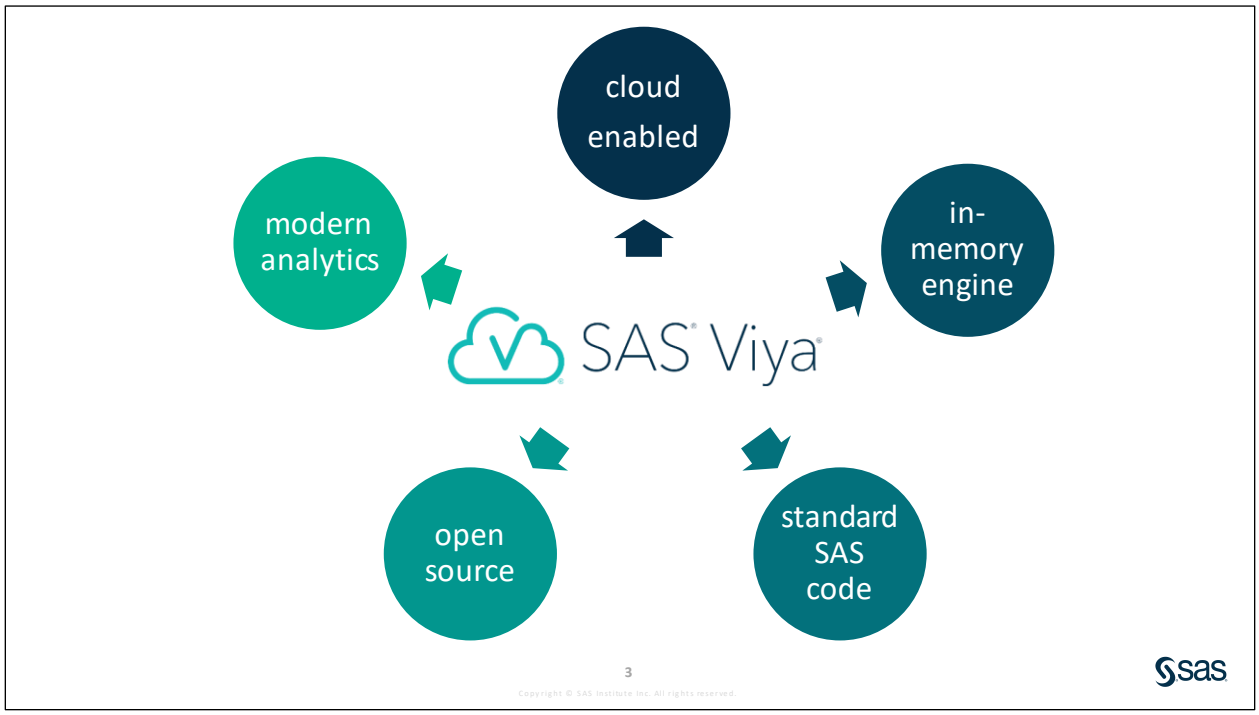
To learn more…

For information about other courses in the curriculum, contact the SAS Education Division at 1-800-333-7660, or send e-mail to training@sas.com. You can also find this information on the web at http://support.sas.com/training/ as well as in the Training Course Catalog.

For a list of SAS books (including e-books) that relate to the topics covered in this course notes, visit https://www.sas.com/sas/books.html or call 1-800-727-0025. US customers receive free shipping to US addresses.

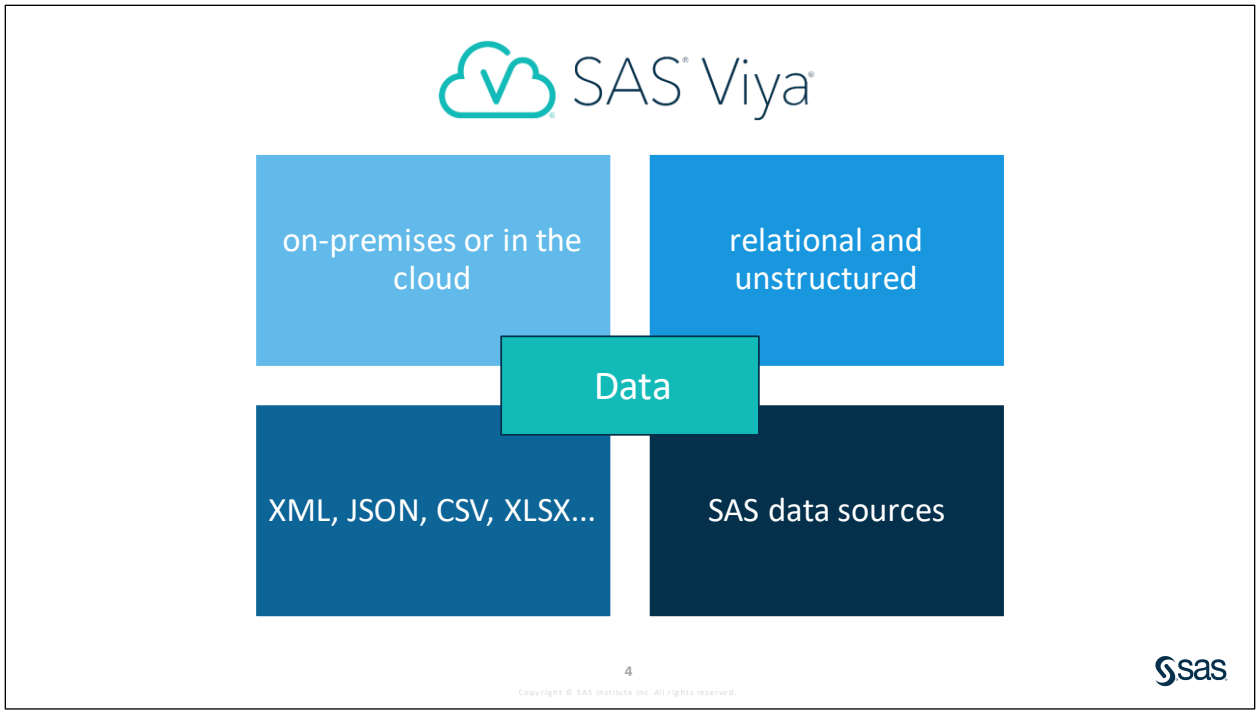
Lesson 1 Introduction

1.1 SAS Viya Overview and Course Setup

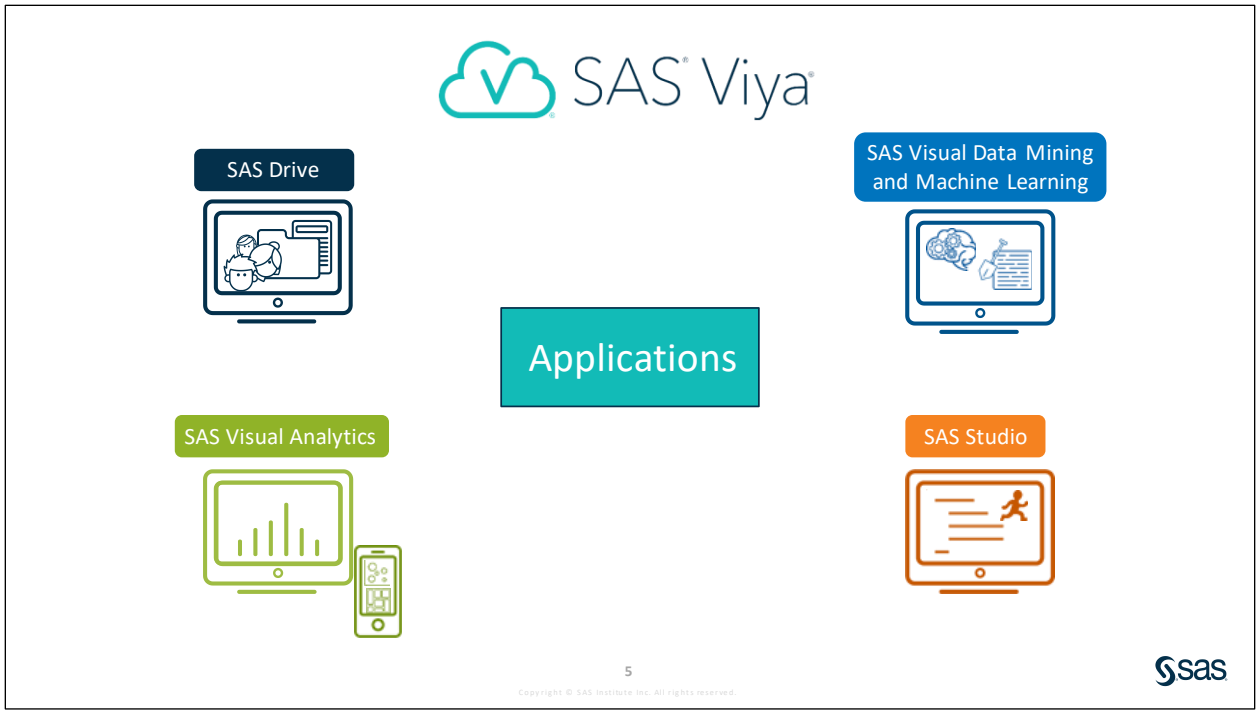


SAS Viya is the latest enhancement of the SAS Platform.

- It is cloud enabled, allowing scalable, web-based access for your particular data processing needs.
- It includes an in-memory engine and parallel processing capabilities that can have a big impact on execution speed.
- Familiar SAS code that you are accustomed to writing in SAS®9 will still work in SAS Viya. However, with a few modifications, you can take advantage of the in-memory processing in SAS Viya for accelerated performance.
- It provides integration with open-source tools like Python, R, and other open-source languages.
- Together, these features in Viya provide a modern analytics platform that can take you from data access and preparation to relevant, actionable results.

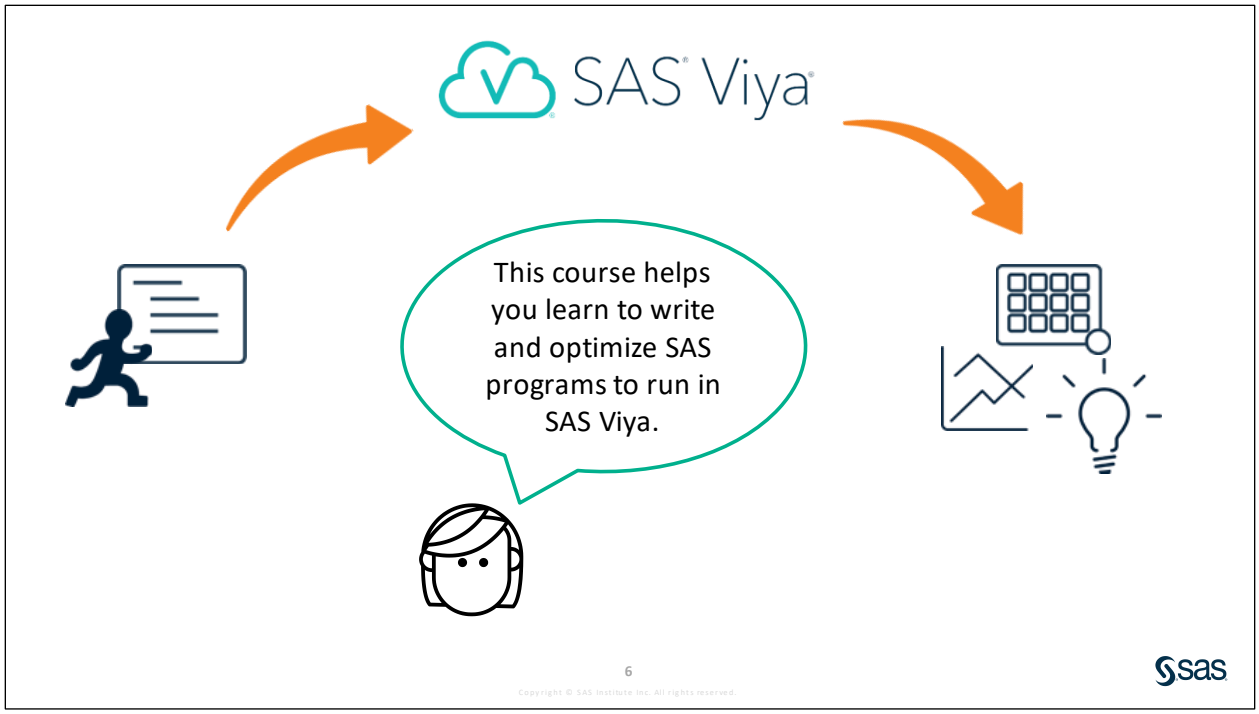


In SAS Viya, you can access a variety of data sources. In addition to SAS tables, you can access data that is on-premises or in the cloud. Data can be in relational databases or unstructured data. Viya can also use other familiar file formats such as XML, JSON, CSV, or XLSX.



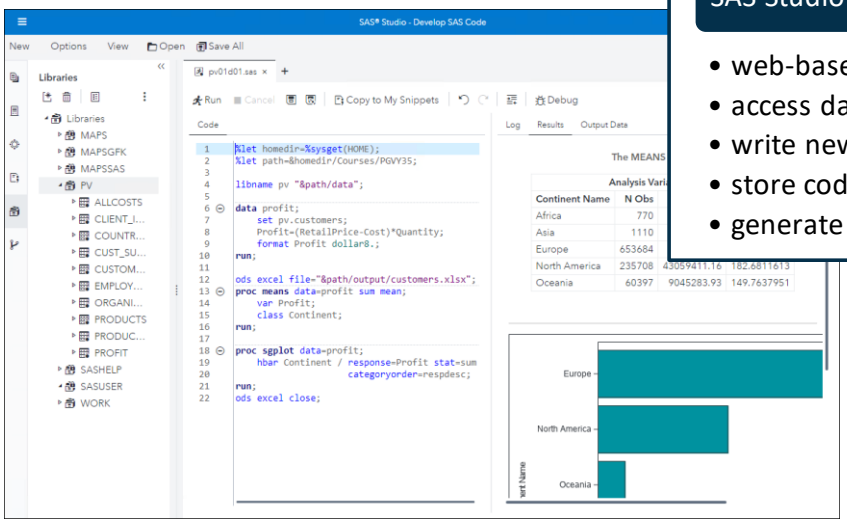
SAS Viya includes several services and applications. Here are a few of these applications:

- SAS Drive, for organizing and accessing all SAS content
- SAS Visual Analytics for web-based reporting and dashboards
- SAS Visual Data Mining and Machine Learning
- SAS Studio for writing and submitting SAS code



Our main focus in this course is learning how to write and optimize SAS programs to run in SAS Viya. We assume that you have a basic knowledge of the SAS programming language for the SAS® 9 Platform.

SAS Viya Programming Interface

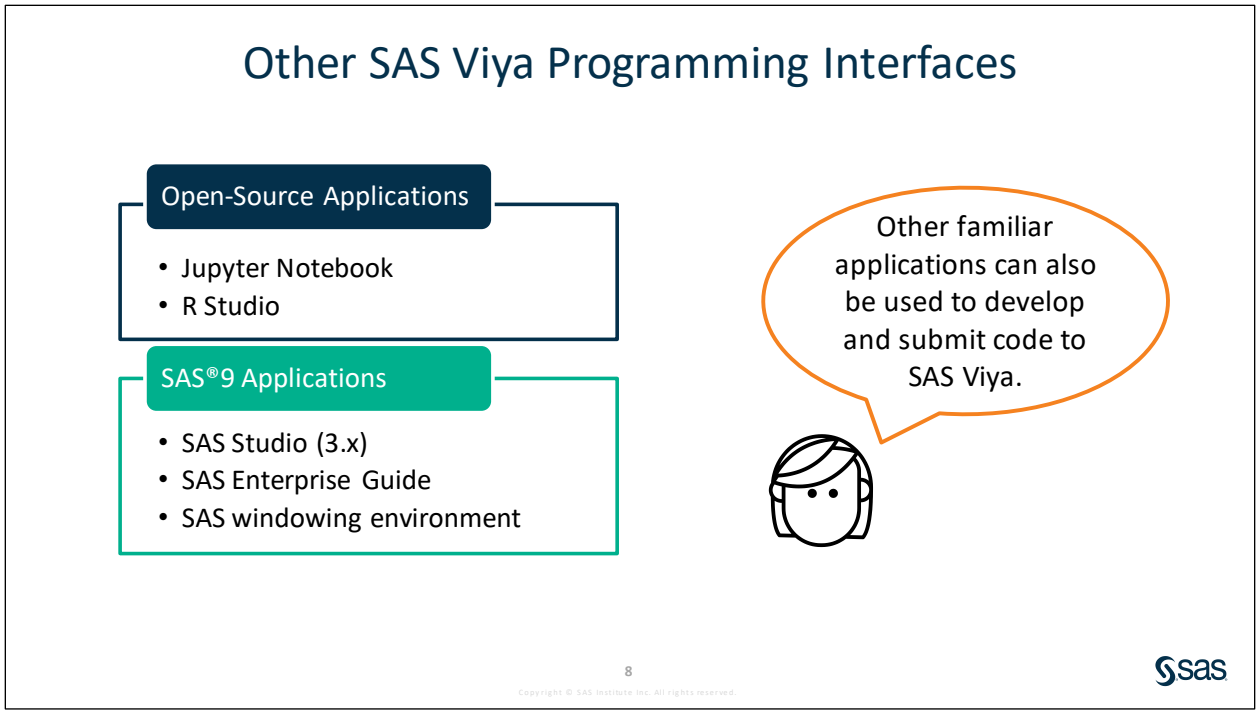


SAS Studio 5.2

- web-based application
- access data and programs
- write new programs
- store code for common actions
- generate code with tasks

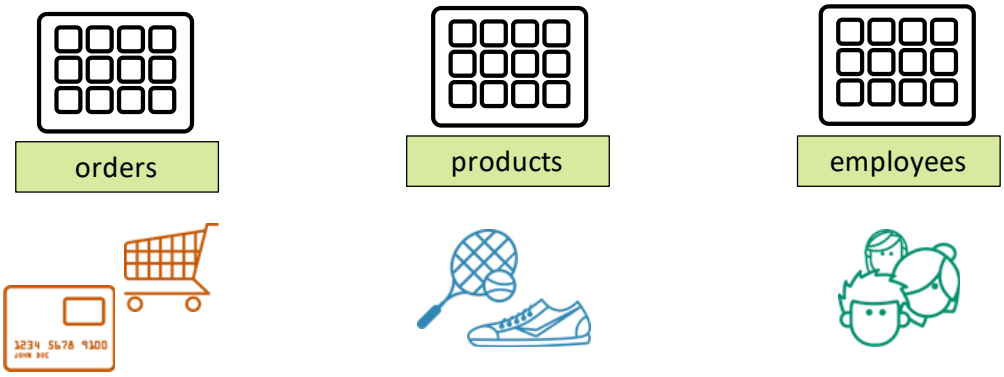
7

SAS Studio is the default interface to write and submit SAS programs in Viya. SAS Studio is a web-based application that includes a variety of helpful features for developing SAS code. In addition to a modern program editor for writing code and viewing the log and results, you can also access files on the local machine or SAS servers, take advantage of stored code for common actions, and generate code with point-and-click tasks. SAS Studio 5.2 is specifically designed for use in SAS Viya.



Although SAS Studio is the default programming interface for Viya, it is not your only option for developing and submitting code. You can use open-source applications, including Jupyter notebook or R Studio. Or if your SAS Platform includes SAS 9.4M5 or later, you can use SAS®9 applications, including SAS Studio 3, SAS Enterprise Guide, or the SAS windowing environment. SAS Studio is both a Viya and SAS®9 application, but it is important to realize that there are different versions. SAS Studio 5 is included with Viya, and SAS Studio 3 is included with SAS®9.

Data Used in This Course



9

Our main focus is learning how to modify and optimize SAS programs to run in SAS Viya. We use simple data from a fictitious retail company that sells sporting goods. Tables include information about customer orders, the products available, and employees.

Practicing in This Course

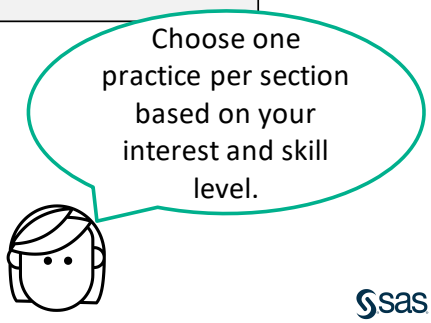
Demonstration	Performed by your instructor as an example for you to observe
Activity	Short practice opportunities for you to perform in SAS, either independently or with the guidance of your instructor
Practice	Extended practice opportunities for you to work on independently
Case Study	A comprehensive practice opportunity at the end of the class

10

As you go through the course, you have different opportunities to practice. You can watch the demos or you can follow along.

Choosing a Practice Level

Level 1	Solve basic problems with step-by-step guidance
Level 2	Solve intermediate problems with defined goals
Challenge	Solve complex problems independently with SAS Help and documentation resources



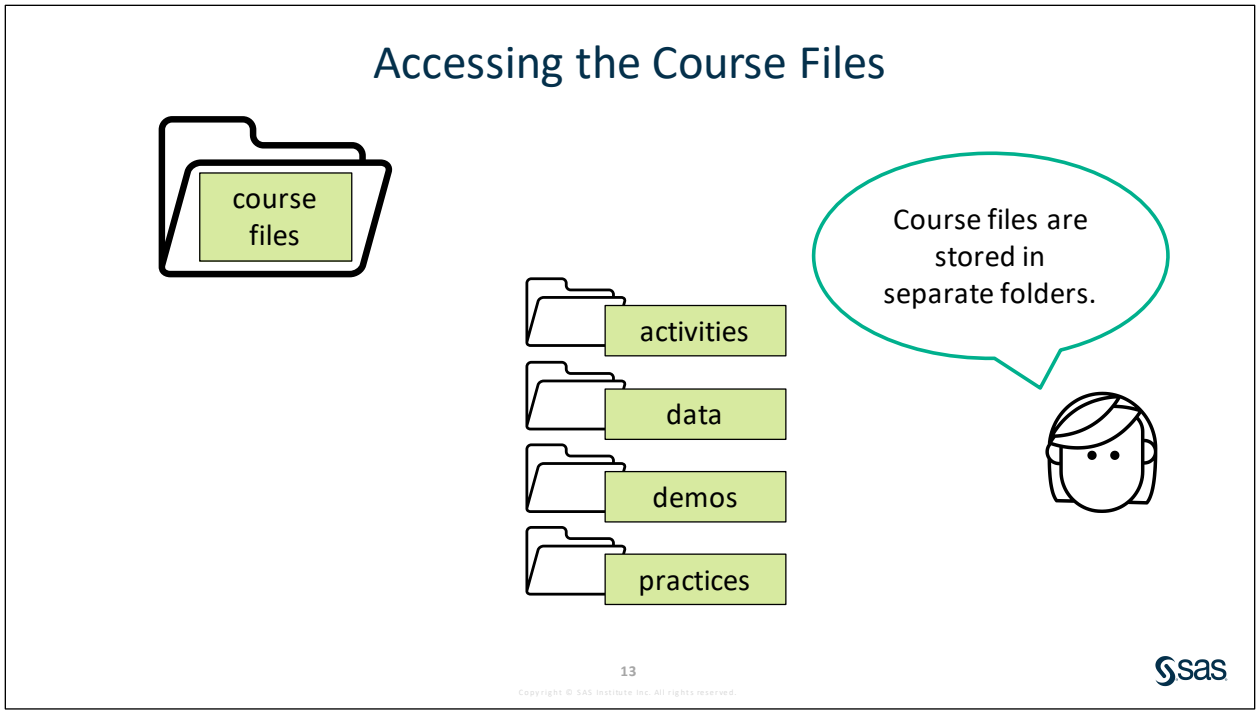
11

When you come to a practice, you choose a level based on your interest and skill level.

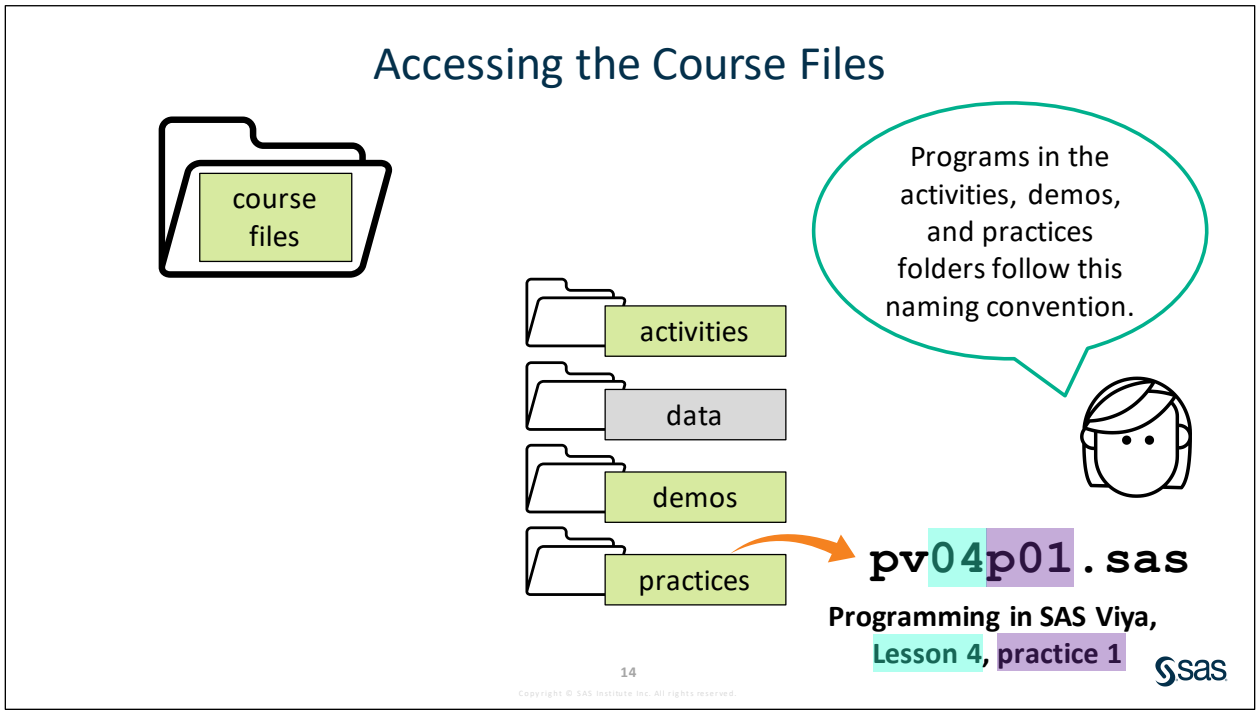
Extending Your Learning

12

Be sure to visit the Extended Learning page for this course if you have not done so already. It contains links to download the course notes and data, instructions to access practice software during your course, and many other resources, including videos, papers, a case study, and other supplemental information.

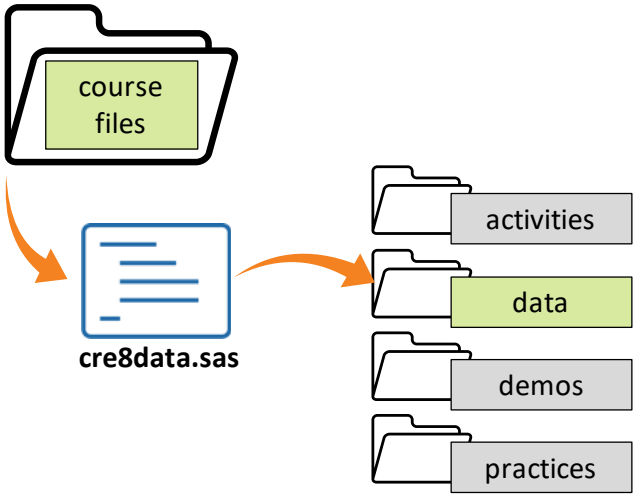


For this course, you use a variety of data files and SAS programs. The SAS program files and data are organized into several folders.



The class data root folder contains subfolders that organize the SAS programs used during class. Program file names follow this convention: the name starts with pv (for Programming in SAS Viya) followed by two digits indicating the lesson number, a letter (a, d, or p) indicating intended use (activity, demo, or practice), and a two-digit serial number. Solution files have the same name as the associated practice or demo file with the letter **s** appended. When you need a file to complete an activity, demo, or practice, the instructions indicate which file you need to open.

Creating the Course Data



15

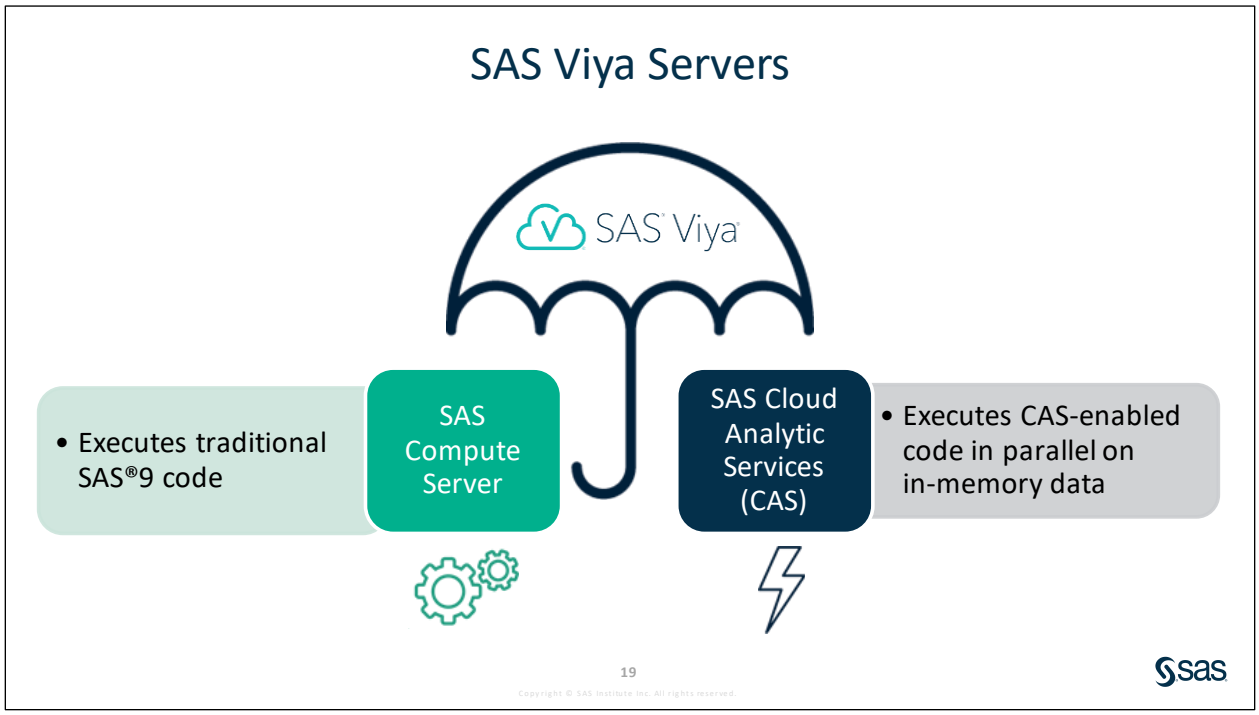
In the next activity, you run a SAS program that creates the SAS data that you need for your environment.

1.01 Activity (Required)

1. Sign in to SAS Viya and open SAS Studio.
   Note: Refer to the access instructions on the Extended Learning page for details about how to sign in.
2. The left side of SAS Studio is the navigation pane. Access the Explorer section of the navigation pane by clicking the icon.
3. Expand <***server***> ⇨ **Home** ⇨ **Courses** ⇨ **PGVY35**. Double-click **cre8data.sas** to open the program.
4. Click **Run** to execute the program. When the program finishes executing, a report appears in the Results tab. View the report and confirm that four files are listed.

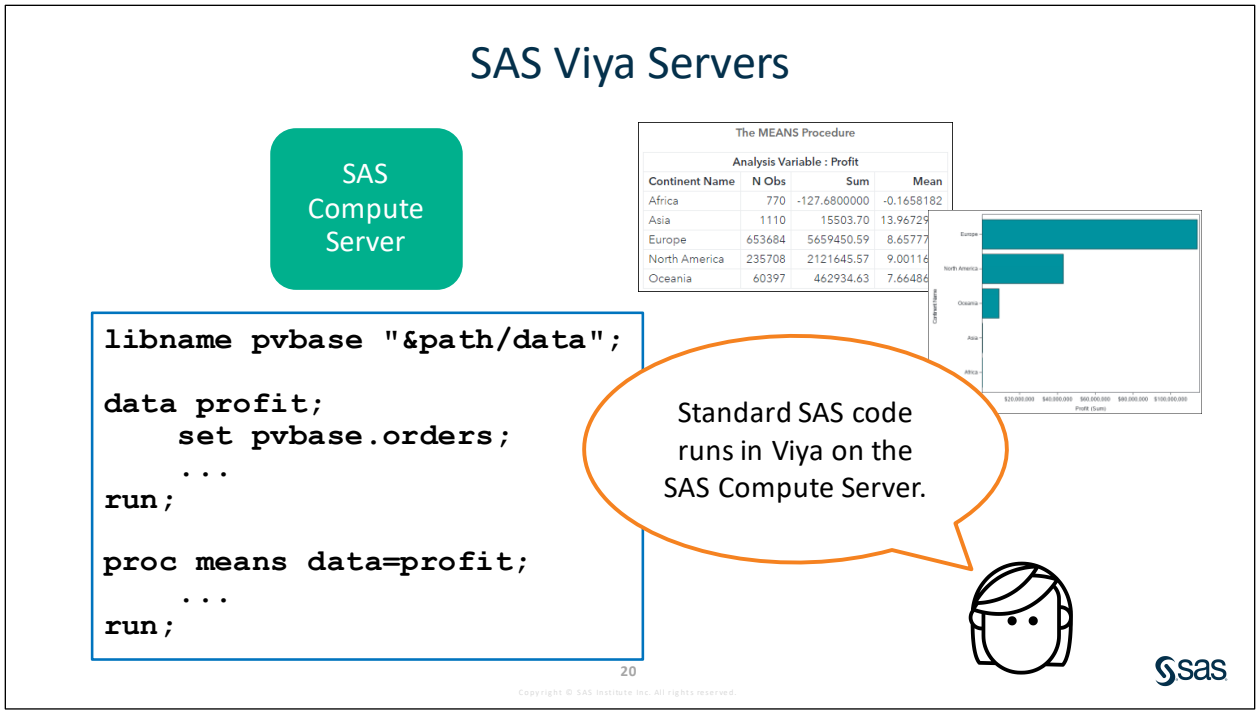
16

1.2 SAS Viya Servers



SAS Viya includes multiple servers to execute SAS code. The two primary servers are the SAS Compute Server and SAS Cloud Analytic Services, or CAS. You can think of the SAS Compute Server as equivalent to the SAS®9 workspace server. Your familiar SAS®9 programs can be submitted as is to SAS Viya, and by default they will run on the Compute Server. There is no need to learn new syntax to use the Compute Server.

SAS Cloud Analytic Services is the high-performance server that allows for parallel processing on in-memory data. It is likely what you will use for your big data and complex analytics. Often, only very minor code modifications are required for programs to run in CAS. Learning how to write or modify SAS programs to run in CAS is the focus of this course.



Notice that this sample program includes a LIBNAME statement, a DATA step, and a PROC step. It can be submitted as is in Viya and will automatically execute on the SAS Compute Server.

Running a SAS Program in SAS Studio

Scenario

Use SAS Studio to open a SAS program, submit it, and navigate the log and results.

Program

- **pv01d01.sas**

Notes

- Standard code that would run in SAS®9 will run in SAS Viya on the Compute Server.

Demo

1. Sign in to SAS Studio.

   Note: Instructions for connecting to SAS Viya and opening SAS Studio are included on the Extended Learning page for this course. You connect to Viya either through a cloud-hosted server or SAS Viya for Learners. Refer to the appropriate *SAS Software Access Instructions* documents in the **Course Materials** section.

2. The navigation pane is on the left side of SAS Studio. Multiple sections in the navigation pane provide access to various resources.

The Open Files section enables you to quickly view and access all of the files that you have opened in your current SAS Studio session.

The Explorer section enables you to access files and folders from your folder shortcuts, your server file system, and your SAS Content Server locations.

The Tasks section enables you to access tasks in SAS Studio. Tasks are based on SAS procedures and generate SAS code and formatted results for you.

The Snippets section enables you to access your saved snippets. Snippets are lines of commonly used code or text that you can save and reuse.

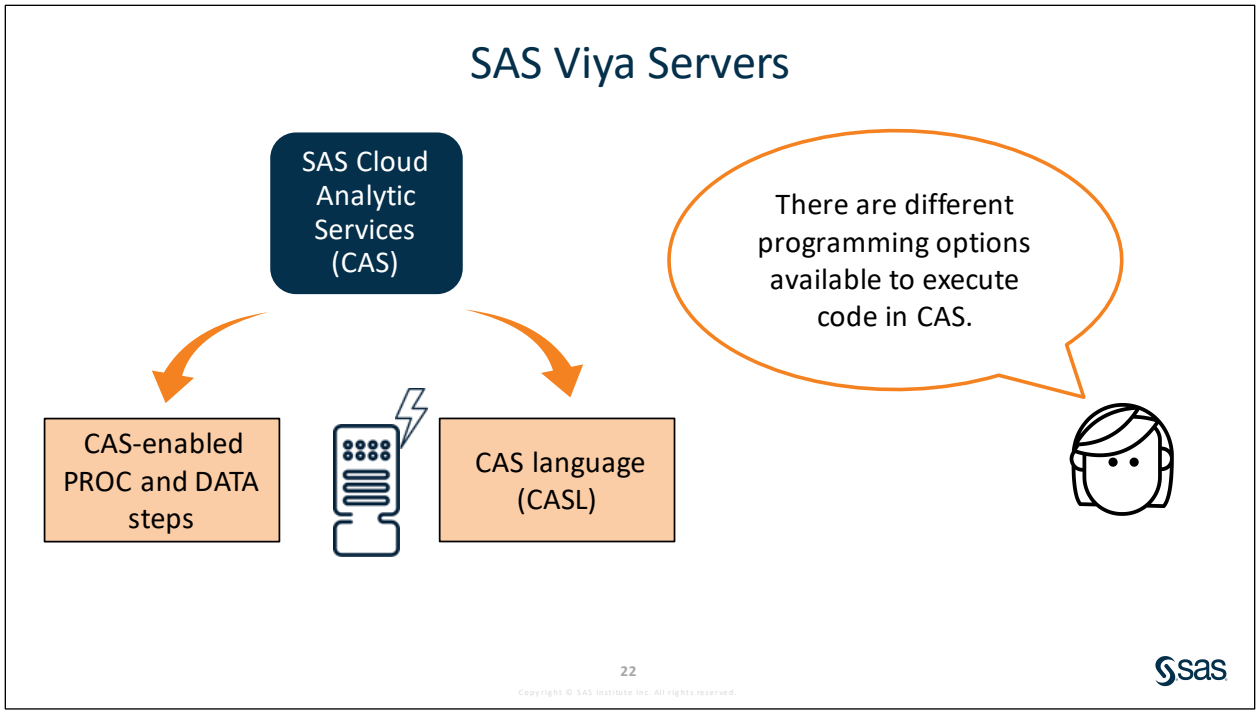
The Libraries section provides access your SAS libraries. You can open SAS tables and add them to your programs. You can expand a table and view the columns in that table.

The Git Repositories section enables you to access the basic Git features from within SAS Studio.

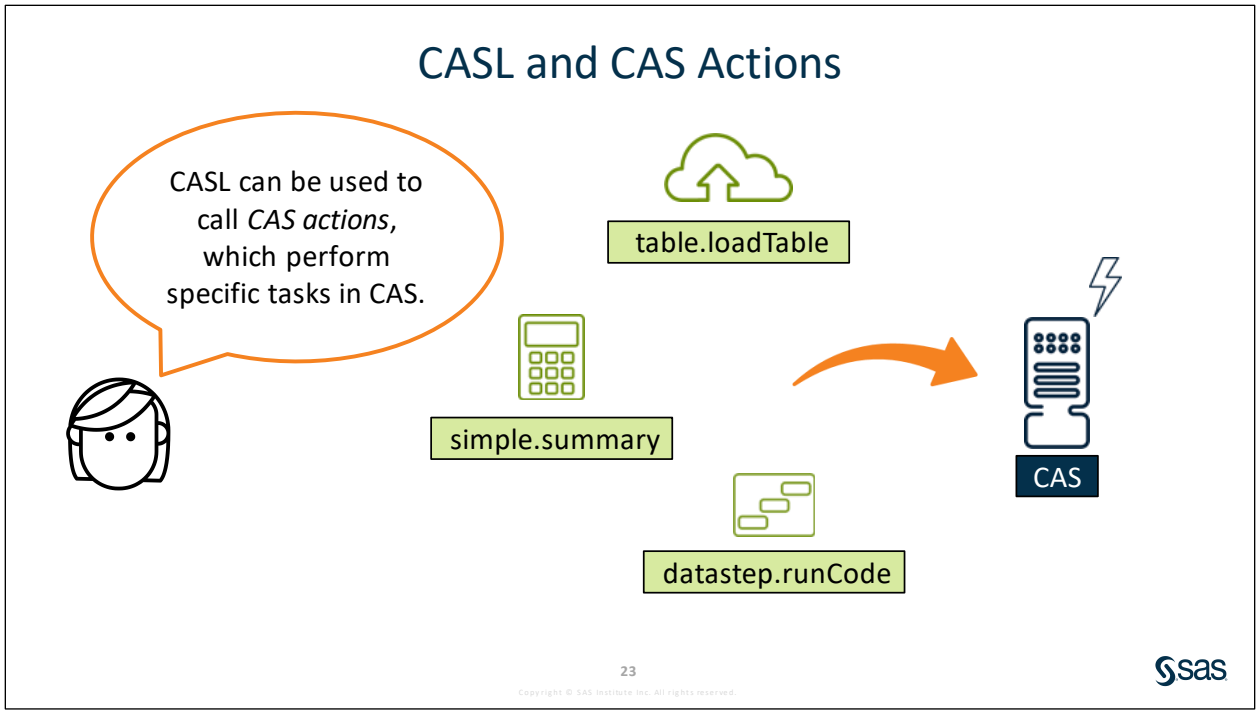
3. Click the **Explorer** icon  in the navigation pane. Expand **<*server*>** ⇨ **Home** ⇨ **Courses** ⇨ **PGYV35** ⇨ **demos**. Double-click the **pv01d01.sas** program to open it.
4. Notice that this program includes DATA and PROC steps, SAS macro code, and ODS statements. Click **Run** to execute the program or press F3.
5. The Code, Log, Results, and Output Data tabs are all available in separate tabs. To change the layout of the tabs, click **More Options**  in the upper right corner of the Program tab. You can select **Single**, **Vertical split**, or **Horizontal split**.

   Note: To return to the single tab view, select **More Options** ⇨ **Tab layout** ⇨ **Single**.
6. Click the **Log** tab. A log summary is available at the top of the window, enabling you to view a listing of messages based on the type. You can click on any of the messages in the log summary to find the corresponding note, warning, or error in the complete log.
7. Click the **Results** tab. SAS Studio produces and displays HTML output by default. The program also generated a Microsoft Excel file in the results folder. In the Explorer section, expand **<*server*>** ⇨ **Home** ⇨ **casuser**. Right-click **customer_report.xlsx** and select **Download file**. The XLSX file is downloaded in your browser, and you can click the downloaded file to open the results in Excel. After viewing the results, close Excel.
8. The Explorer section can also be used to perform file operations. Right-click **customer_report.xlsx** in the **casuser** folder and select **Delete**. Click **Delete** when asked to confirm.
9. Close the **pv01d01.sas** program tab.

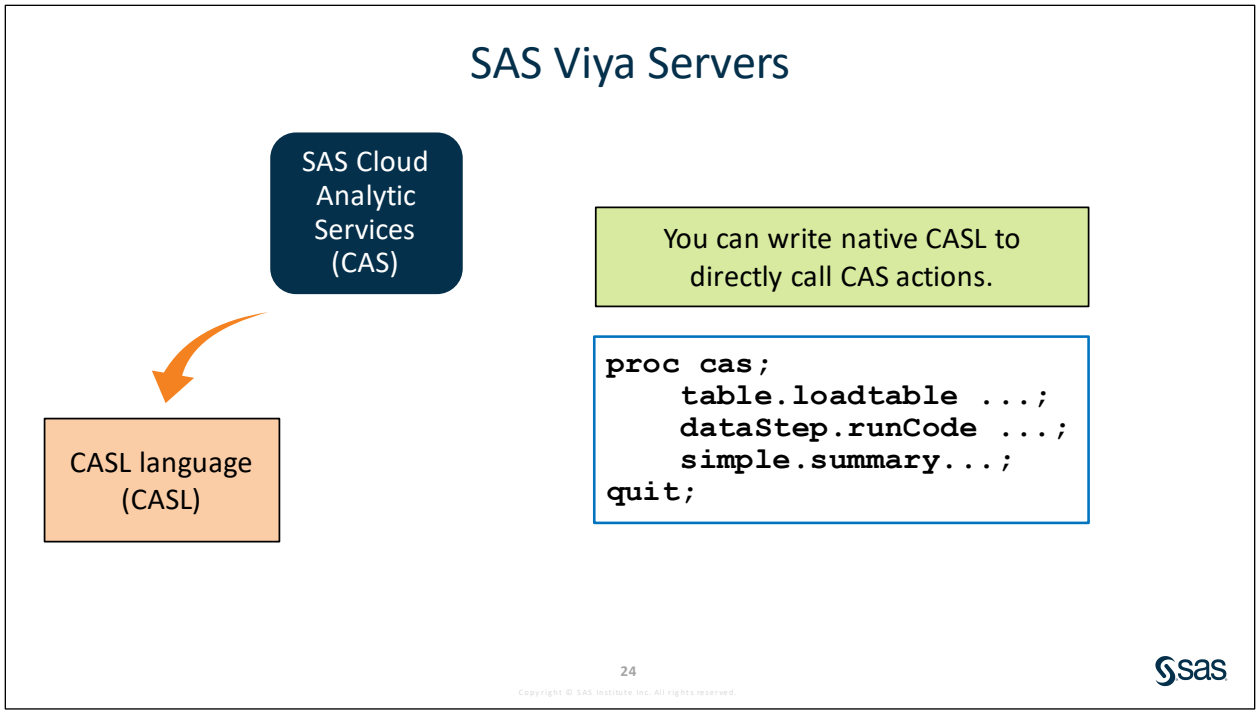
End of Demonstration



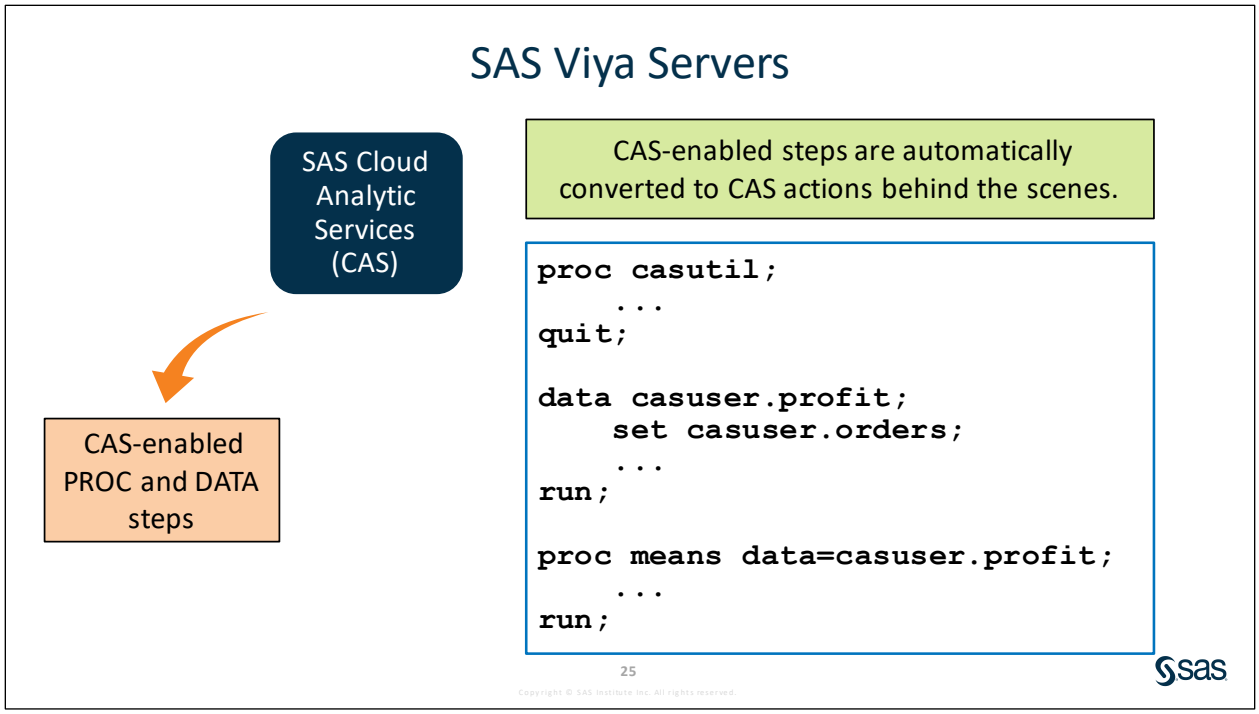
There are different ways to submit requests to CAS for in-memory parallel processing. You can use PROC and DATA steps that are CAS enabled, or you can learn the native CAS language, which is called CASL.



CASL is a scripting language that is used to call CAS actions. CAS actions perform specific tasks. For example, there are actions to load a table to memory, compute summary statistics, or run DATA step code. Many other actions are available to transform data, perform analytics, and create output. Actions that perform similar tasks are grouped into action sets. CASL runs only in CAS, not on the Compute Server.



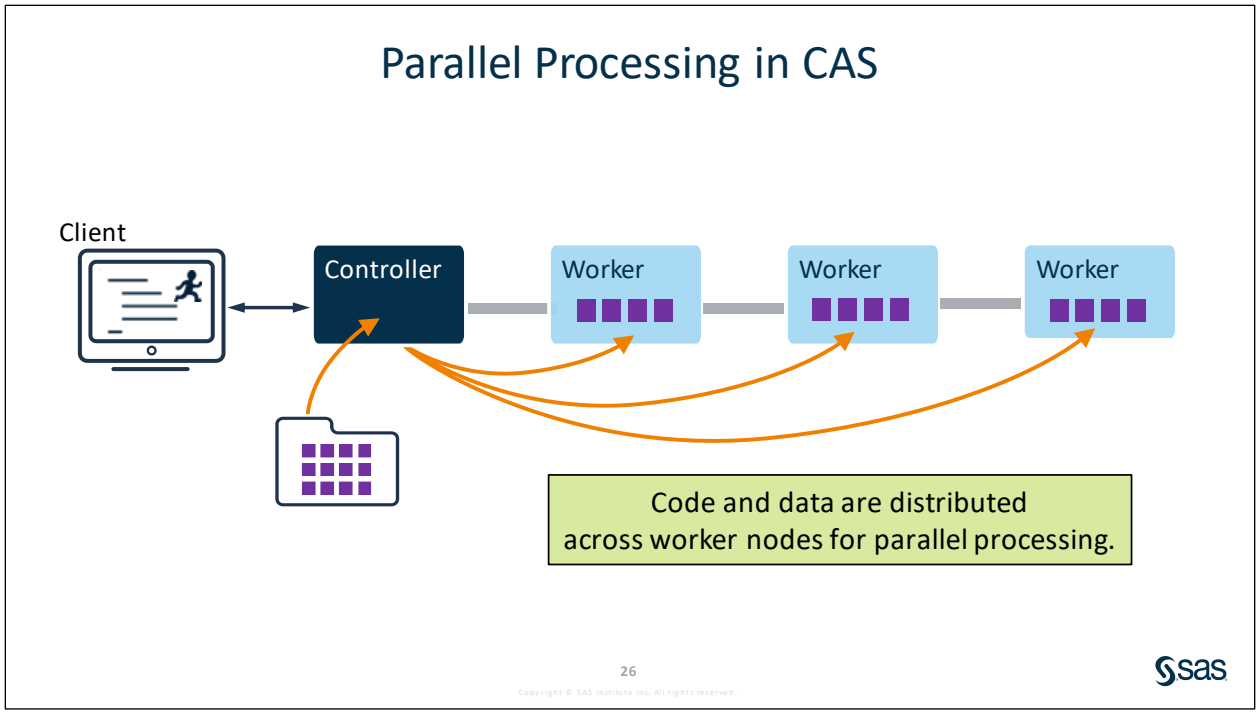
In SAS, you submit CAS actions using PROC CAS and CASL statements. You can also call CAS actions using open-source languages, including Python, Lua, Java, and REST APIs. CASL is not covered in this course, but there are many helpful resources included on the Extended Learning page to get you started.



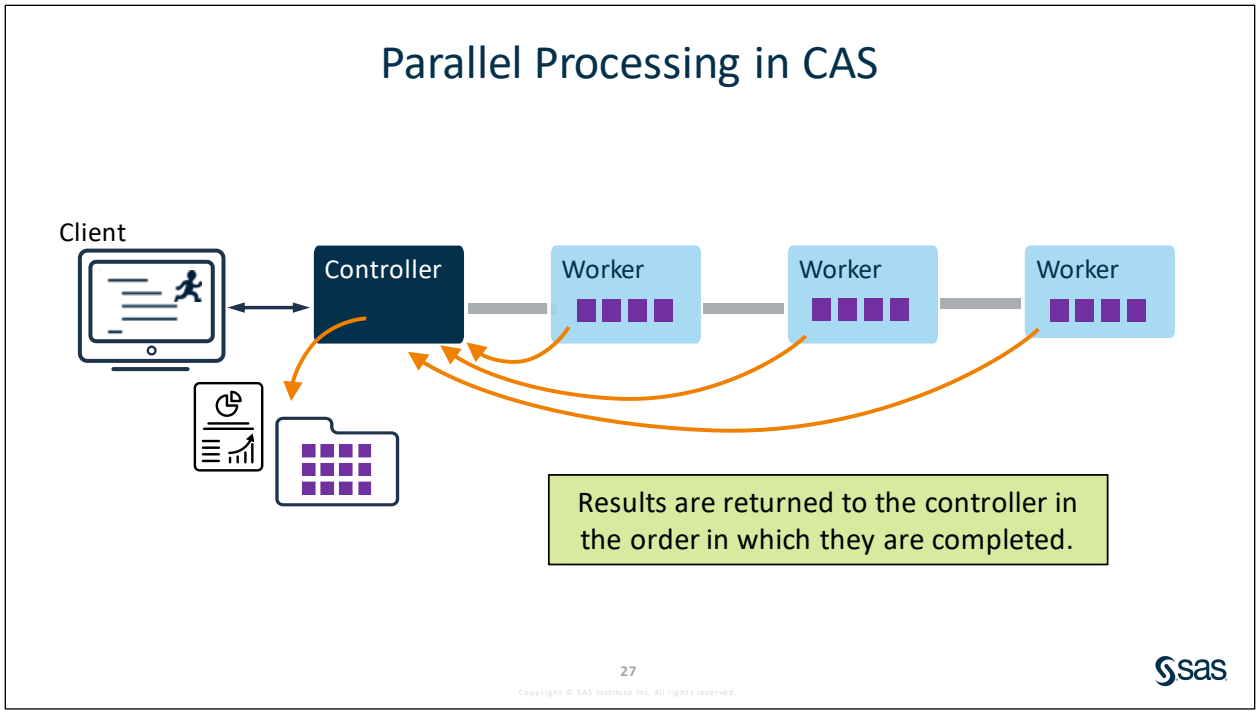
It is not necessary to learn CASL to process data in CAS. Some Base SAS procedures and DATA step features are CAS enabled, meaning that SAS translates compliant code into CAS actions behind the scenes. The luxury of using CAS-enabled statements and steps is that you do not need to learn a new language, but instead you just learn the requirements to ensure that your code is CAS compliant.

DATA steps and PROC steps that are not CAS enabled automatically run on the SAS Compute Server. Our focus in this course is on using CAS-enabled steps and statements.

This program is written to execute in CAS in order to take advantage of parallel processing of in-memory data in the DATA step and PROC MEANS. You will notice that the basic syntax is familiar. However, there are some new statements and steps used to direct processing in CAS.

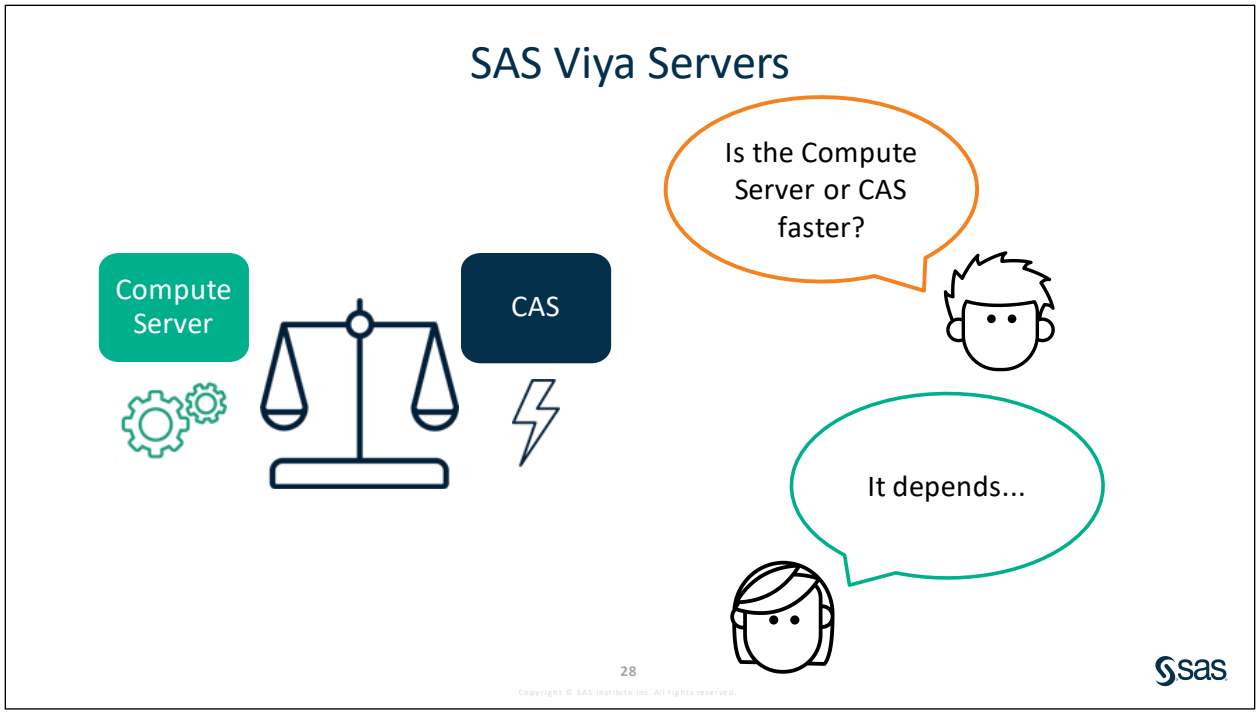


Let's take a high-level look at how CAS works. When a program is submitted, it first goes to the CAS controller. The controller distributes the code and data to separate worker nodes. The nodes perform coordinated parallel processing on their portion of the data. This means that the nodes are executing the same action at the same time.

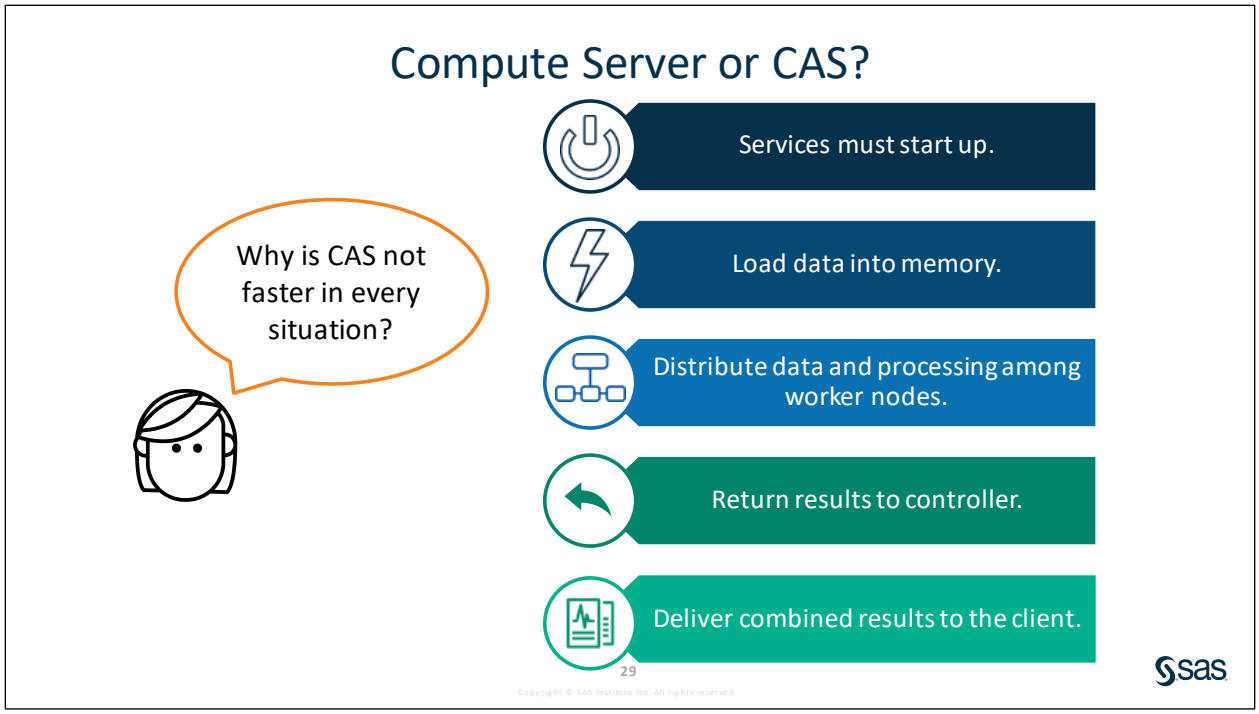


As worker nodes complete processing, they return results back to the controller in the order in which they finish. In this topology, the nodes perform the work much faster than processing the job on one machine.

CAS can be configured to run on a single machine, taking advantage of multiple CPUs or threads to speed up the processing. It can also be configured to run on multiple machines, with one controller node and several worker nodes. Regardless of the configuration, the code that you write is the same, and parallel processing in CAS provides speedy results.



Because CAS performs in-memory parallel processing, does that mean that you should modify all of your code to run in CAS? Probably not. CAS and the Compute Server each have their unique strengths. It depends on the size of your files and the nature of your code as to which steps leverage those strengths. It might be most efficient for your code to execute with a combination of the two servers.

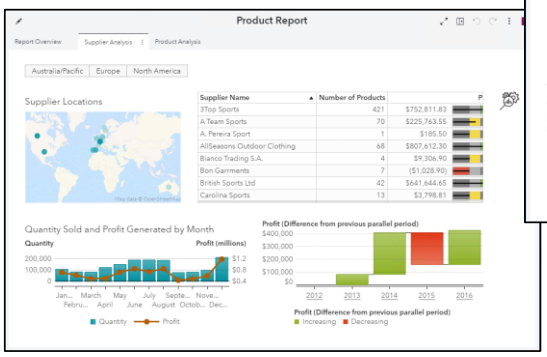


CAS is not always faster than the Compute Server. There is a certain amount of overhead processing that occurs when code runs in CAS. Services must start up, and if the data is not already memory-resident, it must be explicitly loaded into memory. Next, the code required to process the data is distributed by the controller node to the workers. The worker nodes then execute the code and return information to the controller node to combine and deliver the final results.

For large data and complex processes, this overhead might be negligible compared to the accelerated execution time in CAS. For smaller tables and processes, it might be more efficient for the code to run on the Compute Server. To help understand when CAS might be the best option for processing, let's discuss its primary strengths.

Cloud Analytic Services (CAS) Strengths

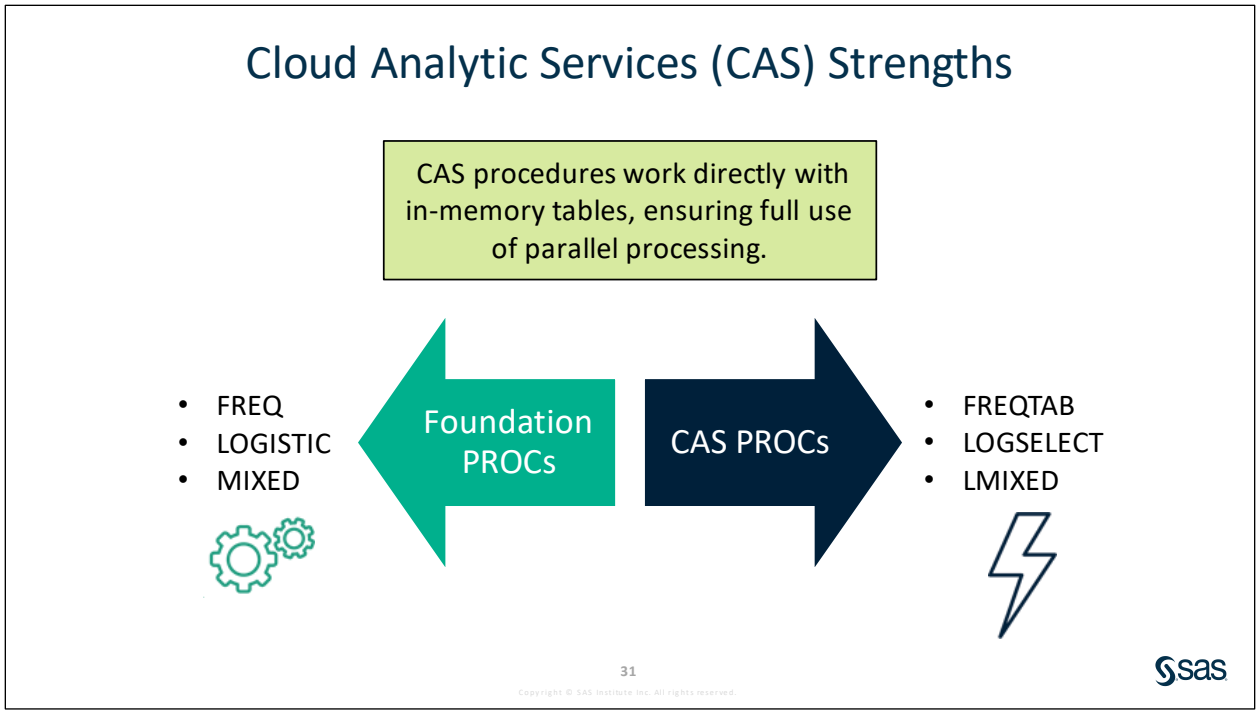
The visual interfaces in Viya all use CAS behind the scenes.



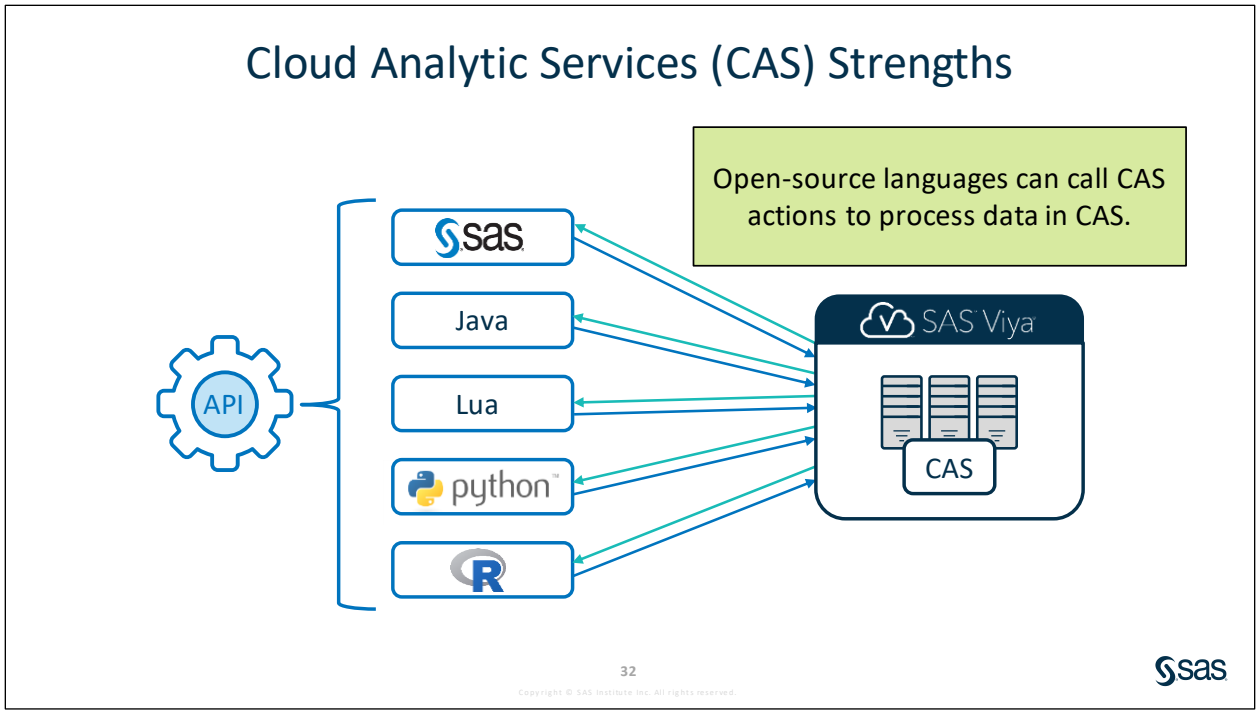
- SAS Visual Analytics
- SAS Visual Statistics
- SAS Visual Data Mining and Machine Learning

30

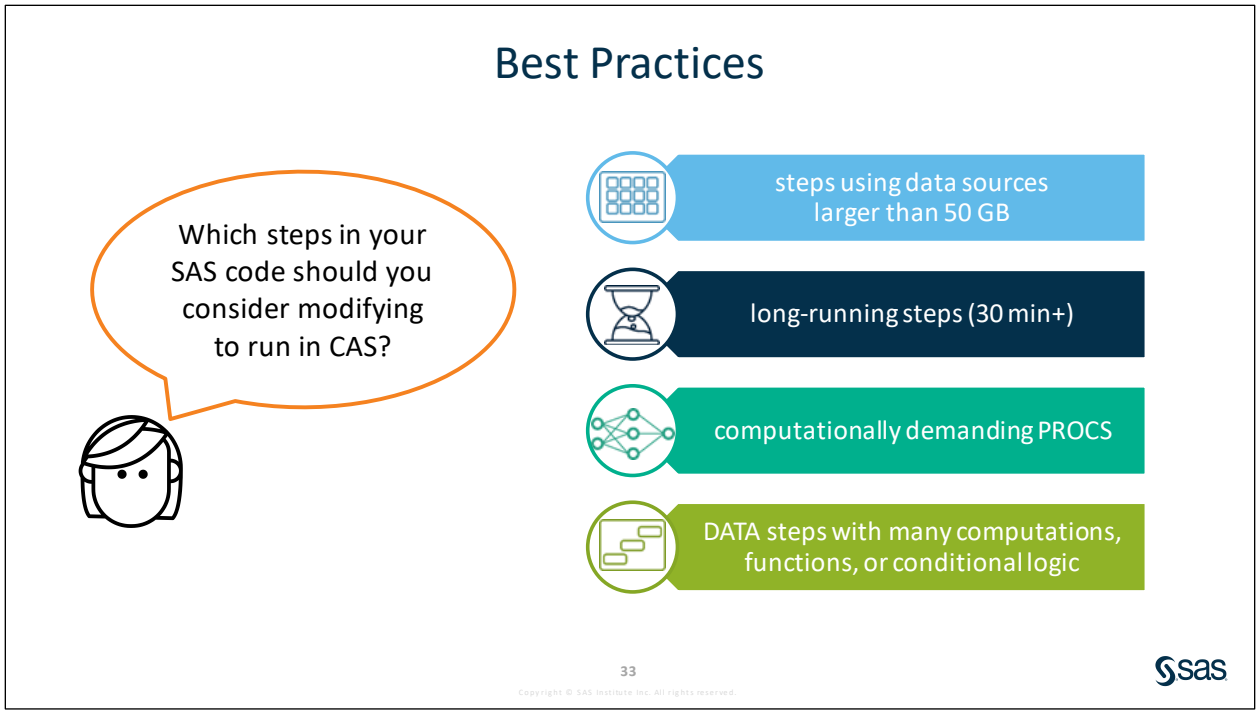
All of the visual interfaces in Viya use CAS behind the scenes, including SAS Visual Analytics, SAS Visual Statistics, and SAS Visual Data Mining and Machine Learning. If you want to leverage any of these applications, you need to process your data in CAS.



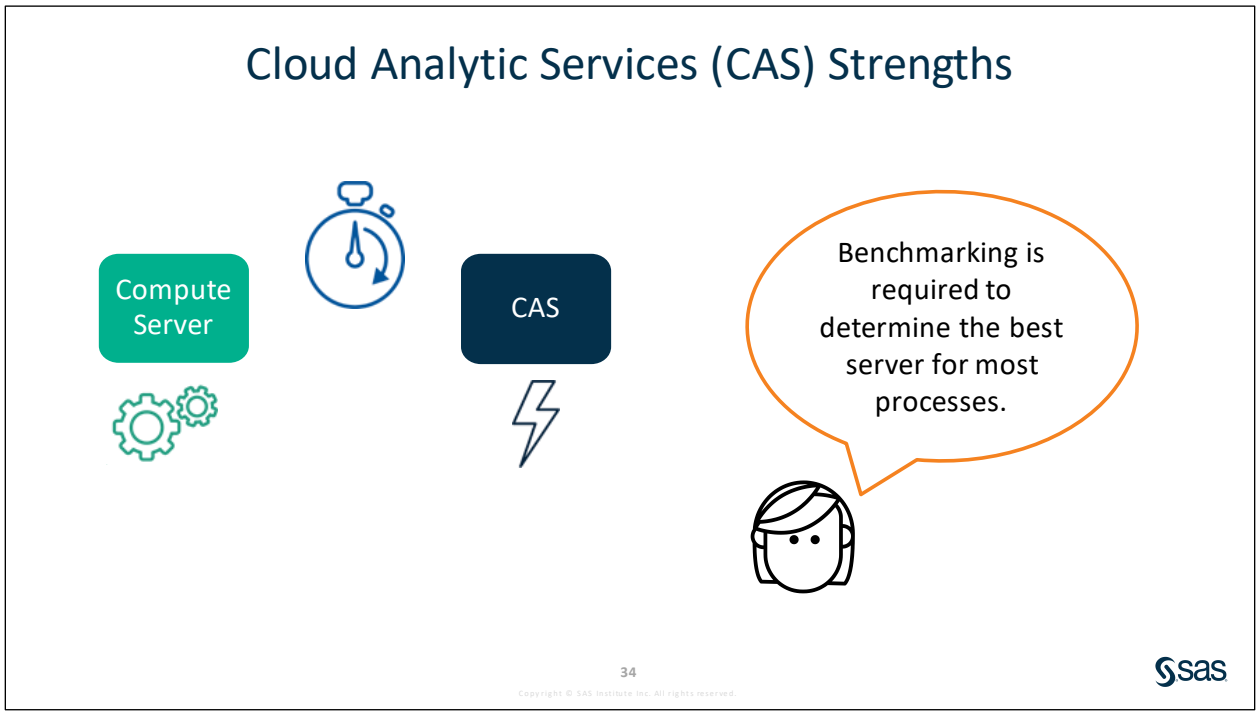
All of the Foundation procedures that you know and love still run on the SAS Compute Server. Some of the Foundation procedures have even been updated in Viya to execute either on the Compute Server or in CAS. But CAS' real strength is the CAS-specific PROCS that have been optimized for parallel, in-memory processing. For example, you can use a PROC FREQ step, but depending on the size and complexity of your data, it can be more efficient to use the equivalent PROC FREQTAB step to process in CAS.



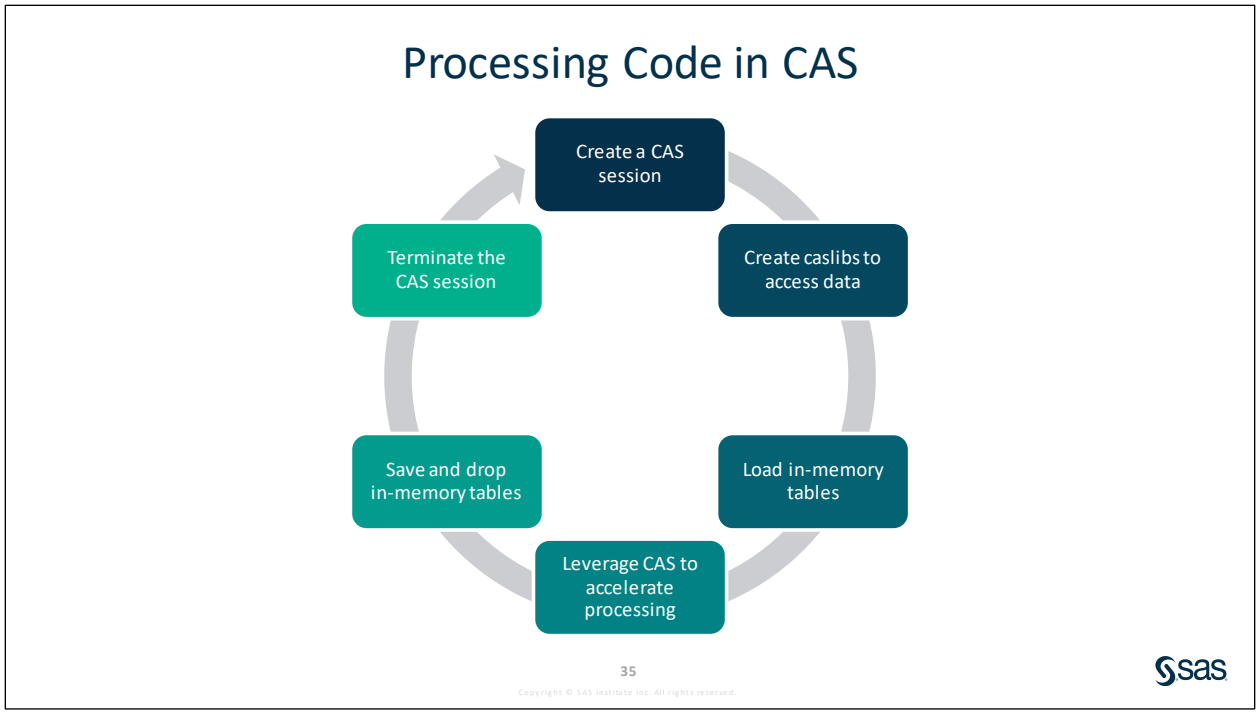
Another strength of CAS is the ability to integrate with common open-source languages. CAS actions can be called directly from Java, Lua, Python, R, or REST APIs. Integrating SAS and open-source technologies is often helpful for two main reasons: programmatically accessing SAS Viya using open-source software, or bringing open-source models into Viya.



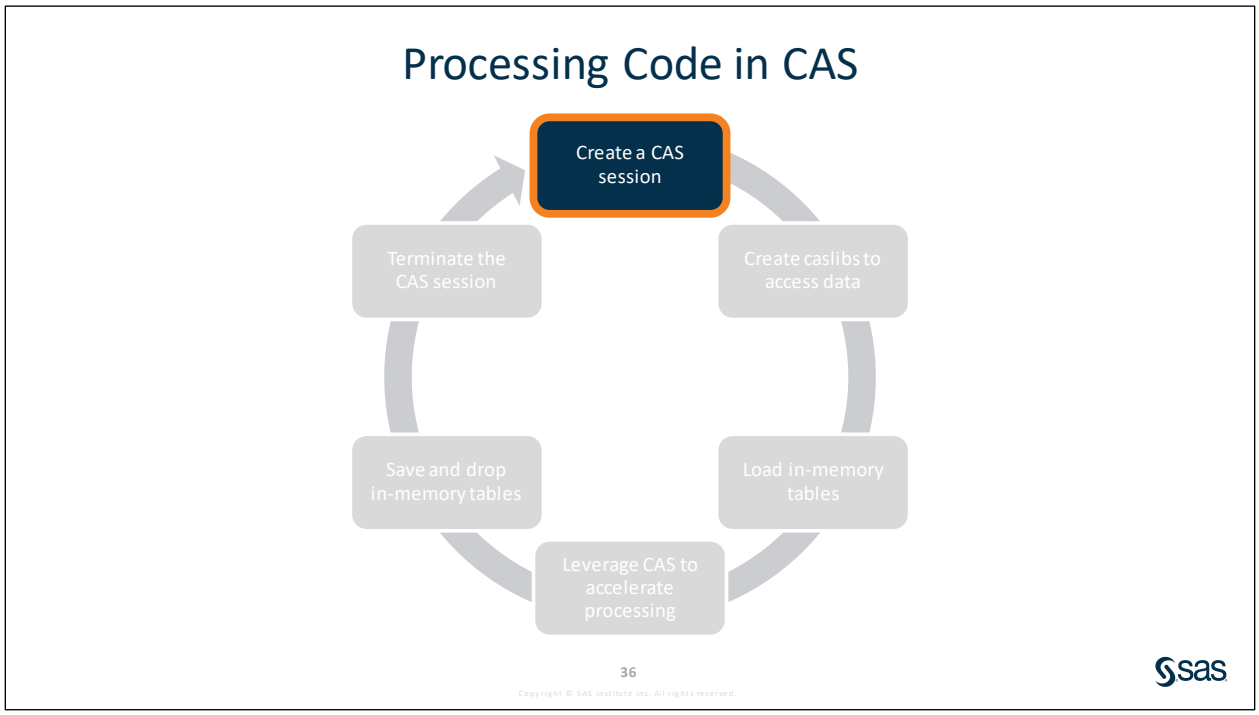
As you work in your Viya environment and consider writing or modifying programs to run in CAS, there are a few best practices to consider. CAS generally outperforms the Compute Server if you have data sources larger than 50 GB, steps that run for 30 minutes or longer, PROCs that are computationally demanding, or DATA steps with extremely long or complex logic.



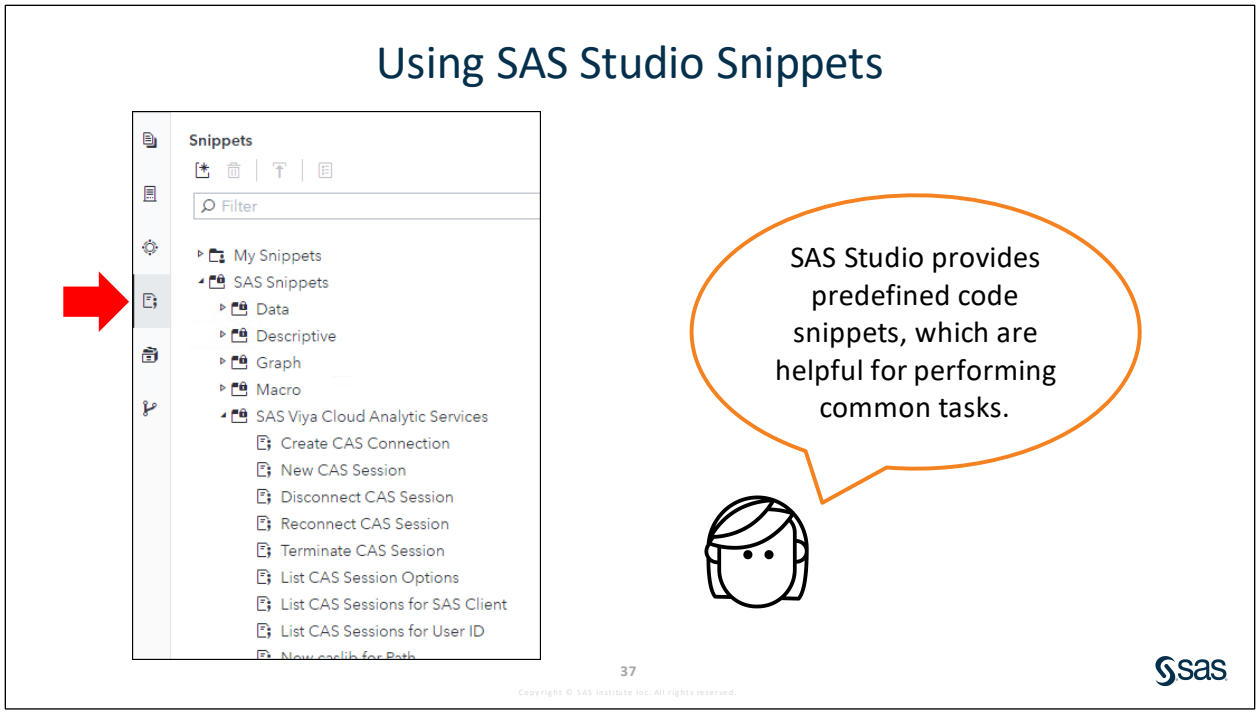
As you become familiar with Viya, comparing processes running in CAS versus the Compute Server will help you to realize the strengths of each.



As we proceed through this course, we use these recommended steps to execute SAS code in CAS. First, create a CAS session then load data files into in-memory tables. Next, leverage CAS where appropriate in programs to improve processing speed. Finally, clean up by releasing in-memory tables and ending your CAS session.



Let's start by creating a CAS session.



We will use SAS Studio *code snippets* to insert SAS code into our programs. There are several snippet categories, but we will primarily use the SAS Viya Cloud Analytic Services snippets.

Some snippets provide complete programs that can be submitted with no modifications required. Other snippets are templates that require additional input before the program can be submitted (such as filling in an option or table name). You can also create your own custom snippets to save code that you use frequently.

Connecting to a CAS Server Using a Snippet

Scenario

Open SAS Studio and start a CAS session using a code snippet.

Syntax

CAS *session-name* <*option(s)*>**;**

Notes

- The CAS statement is used to connect to Cloud Analytic Services. You can open this code directly in SAS Studio in the Snippets section of the navigation pane  . Expand **SAS Snippets** ⇨ **SAS Viya Cloud Analytic Services**, and then double-click **New CAS session**.

```
cas mySession sessopts=(caslib=casuser timeout=1800 locale="en_US");
```

- The CAS session name is **mySession**. If you do not specify a session name, the default session name is **Casauto**.

```
cas;
```

- The SESSOPTS= option enables you to specify one or more session option settings to apply during or after CAS session start-up. If you do not use the SESSOPTS= option, all of the default settings are used.
- You can use a SAS Studio snippet to generate a list of the CAS session options by selecting **SAS Snippets** ⇨ **SAS Viya Cloud Analytic Services**, and then double-clicking **List CAS Session Options**.

Selected Suboptions for the SESSOPTS= Option

Suboption	Purpose
CASLIB=*caslib*	Specifies the active caslib for the CAS session.
TIMEOUT=	Specifies the SAS Cloud Analytic Services session time-out in seconds for a new or existing session.
LOCALE=	Specifies a set of attributes in a SAS session that reflect the language, local conventions, and culture for a geographical region.

Demo

1. Sign in to SAS Studio.

   Note: Instructions for connecting to SAS Viya and opening SAS Studio are included on the Extended Learning page for this course. You connect to Viya either through a cloud-hosted server or SAS Viya for Learners. Refer to the appropriate *Access Instructions* documents in the **Course Materials** section.

2. Access the Snippets section of the navigation pane and expand **SAS Snippets** ⇨ **SAS Viya Cloud Analytic Services**. Right-click **New CAS session** and select **Open**, or double-click to open the snippet.

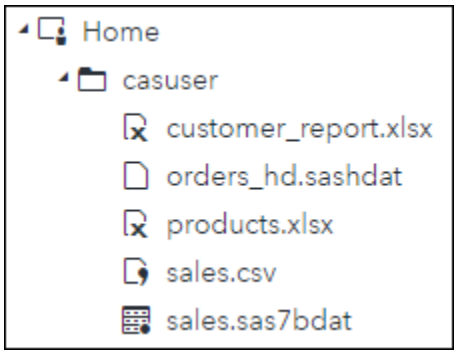
3. Examine the CAS statement.

```
cas mySession sessopts=(caslib=casuser timeout=1800
locale="en_US");
```

4. Submit the program (press the F3 key) to connect to the CAS server session and examine the log to verify that the CAS session **MYSESSION** was successfully started.

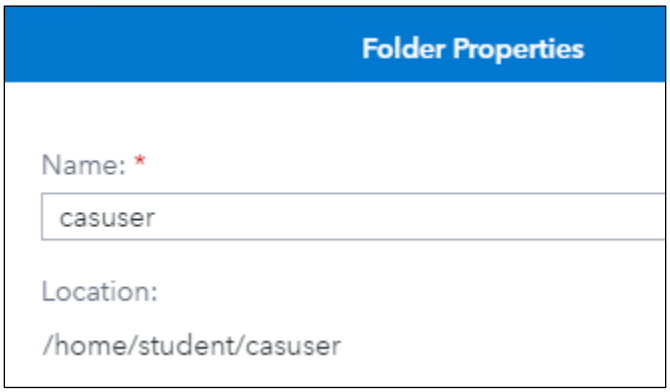
```
NOTE: The session MYSESSION connected successfully to Cloud Analytic Services
server.exnet.sas.com using port 5570. The UUID is f000e610-2a3a-b249-9a7a-645688943631. The
user is student and the active caslib is CASUSER(student).
NOTE: The SAS option SESSREF was updated with the value MYSESSION.
NOTE: The SAS macro _SESSREF_ was updated with the value MYSESSION.
NOTE: The session is using 0 workers.
NOTE: 'CASUSER(student)' is now the active caslib.
NOTE: The CAS statement request to update one or more session options for session MYSESSION
completed.
```

5. Access the Explorer section and expand <***server***> ⇨ **Home** ⇨ **casuser**. This folder includes a subset of the files that we will use in the course. Notice that the files have extensions of .sashdat, .xlsx, .csv, and .sas7bdat.



6. To write a program that accesses files in this folder, you might need the fully qualified path. Right-click the **casuser** folder and select **Properties**. Note that the **Location** field indicates the fully qualified path for this folder. Click **OK** to close the dialog box.

   Note: Your path might differ depending on your environment.



7. To terminate the CAS session, access the Snippets section and expand **SAS Snippets** ⇨ **SAS Viya Cloud Analytic Services**. Then right-click **Terminate CAS Session** and select **Open**, or double-click to open the snippet.
8. Submit the code (F3) to terminate the CAS session.

```
NOTE: Deletion of the session MYSESSION was successful.
NOTE: The default CAS session MYSESSION identified by SAS option SESSREF= was terminated. Use
the OPTIONS statement to set the
SESSREF= option to an active session.
NOTE: Request to TERMINATE completed for session MYSESSION.
```

9. Right-click any of the program tabs and select **Close all**. If you are prompted to save the modified programs, click **Don't Save**.

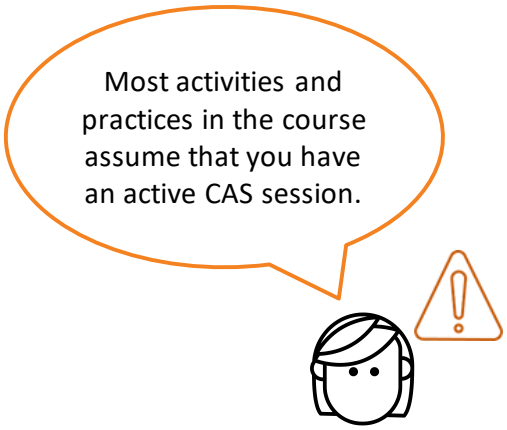
End of Demonstration

1.02 Activity

1. Sign in to SAS Viya.
   Note: Refer to the access instructions on the Extended Learning page for details about how to sign in.
2. In the Snippets section of the navigation pane, expand **SAS Snippets** ⇨ **SAS Viya Cloud Analytic Services**. Double-click **New CAS Session** and submit the snippet code.
3. View the log and confirm that the CAS session connected successfully. What is the name of the session and the default caslib?

39

Create New CAS Session




```
cas mysession;
```

Submit this statement or the **New CAS Session** snippet if your CAS or SAS session ends.

41

1.3 Solutions

Solutions to Activities and Questions

1.01 Activity – Correct Answer

View the report and confirm that four files are listed.

Files loaded to CASUSER

The CASUTIL Procedure

CAS File Information

Name	Permission	Owner	Group	Encryption Method	File Size	Last Modified (UTC)
orders_hd.sashdat	-rwxr-xr-x	cas	sas	NONE	1.6GB	17JUN2020:17:21:22
products.xlsx	-rwxr-xr-x	cas	sas		560.8KB	17JUN2020:17:21:29
sales.csv	-rwxr-xr-x	cas	sas		10.3KB	17JUN2020:17:21:29
sales.sas7bdat	-rwxr-xr-x	cas	sas		72.0KB	17JUN2020:17:21:29

You need to run this program only once to create the data in the appropriate folders.

17

1.02 Activity – Correct Answer

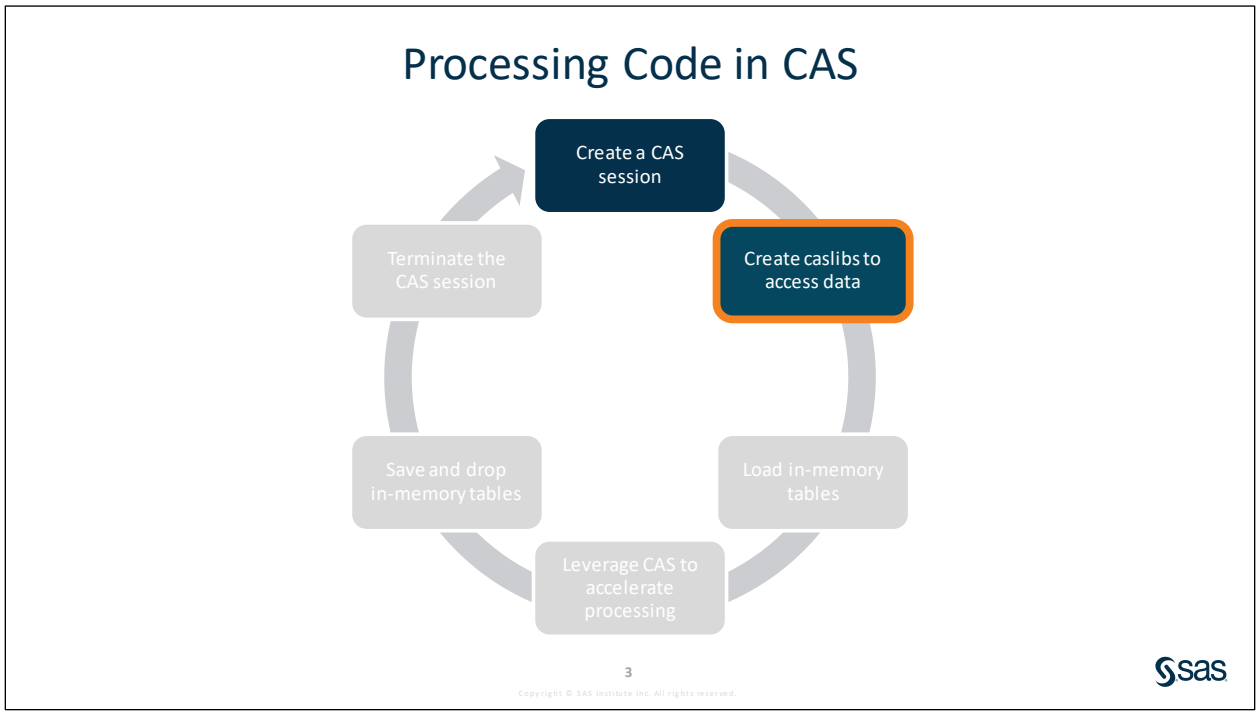
3. View the log and confirm that the CAS session connected successfully. What is the name of the session and the default caslib?
   MYSESSION and CASUSER

```
88 cas mySession sessopts=(caslib=casuser timeout=1800 locale="en_US");
NOTE: The session MYSESSION connected successfully to Cloud Analytic
      Services server.demo.sas.com using port 5570. The UUID is...
      The user is student and the active caslib is CASUSER(student).
NOTE: The SAS option SESSREF was updated with the value MYSESSION.
NOTE: The SAS macro _SESSREF_ was updated with the value MYSESSION.
NOTE: The session is using 0 workers.
NOTE: 'CASUSER(student)' is now the active caslib.
NOTE: The CAS statement request to update one or more session options
      for session MYSESSION completed.
```

40

Lesson 2 Loading Data into SAS® Cloud Analytic Services (CAS)

2.1 Understanding Caslibs



After we start a CAS session, the next step is to load data into memory using a caslib.

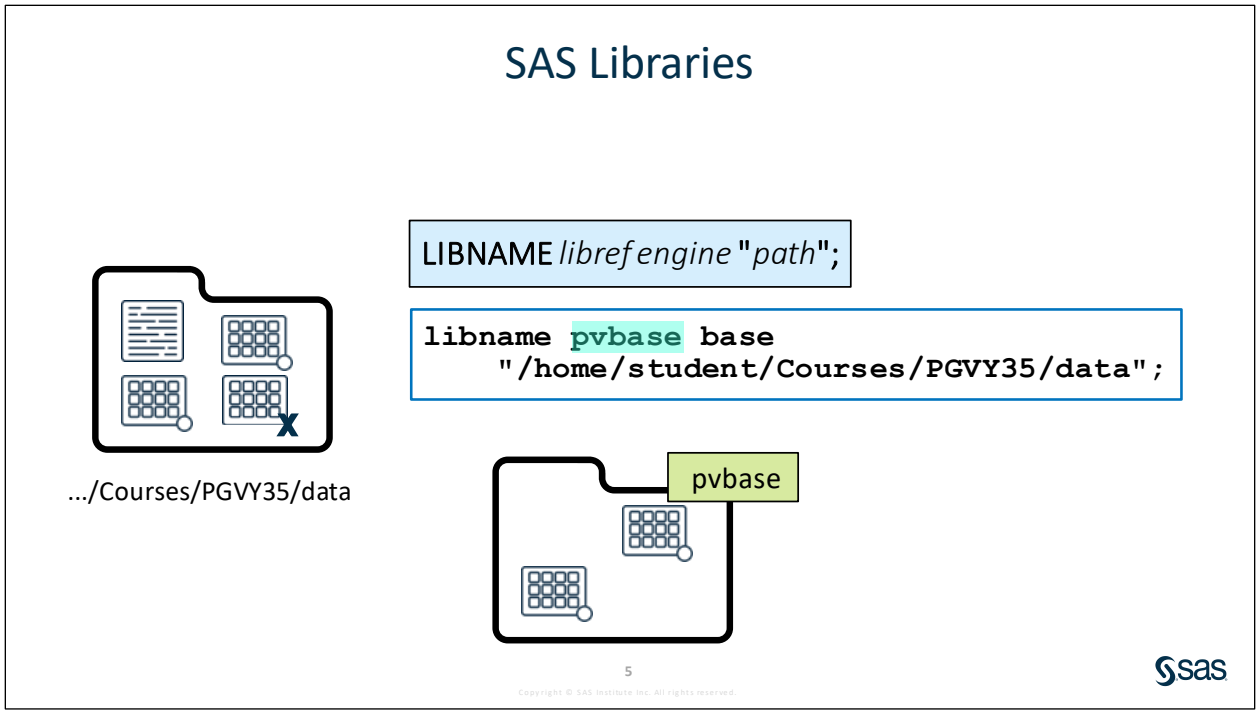
SAS Libraries

LIBNAME *libref engine* "*path*";

.../Courses/PGVY35/data

4

Let's start with what you know about libraries. As a SAS programmer, you are familiar with SAS libraries and the LIBNAME statement. You might have a folder or directory on your local machine that contains various file types including SAS data sets, CSV files, Microsoft Excel files, and more. In order to access the SAS data sets, you associate a SAS library with the folder.



This SAS LIBNAME statement creates the library, **pvbase**, and uses the Base engine to access SAS files within the directory, /home/student/Courses/PGVY35/data. **Pvbase** is a library reference name, or *libref*, that serves as an alias to the physical location. SAS offers additional engines to access other file types, including Excel workbooks and various databases.

2.01 Activity

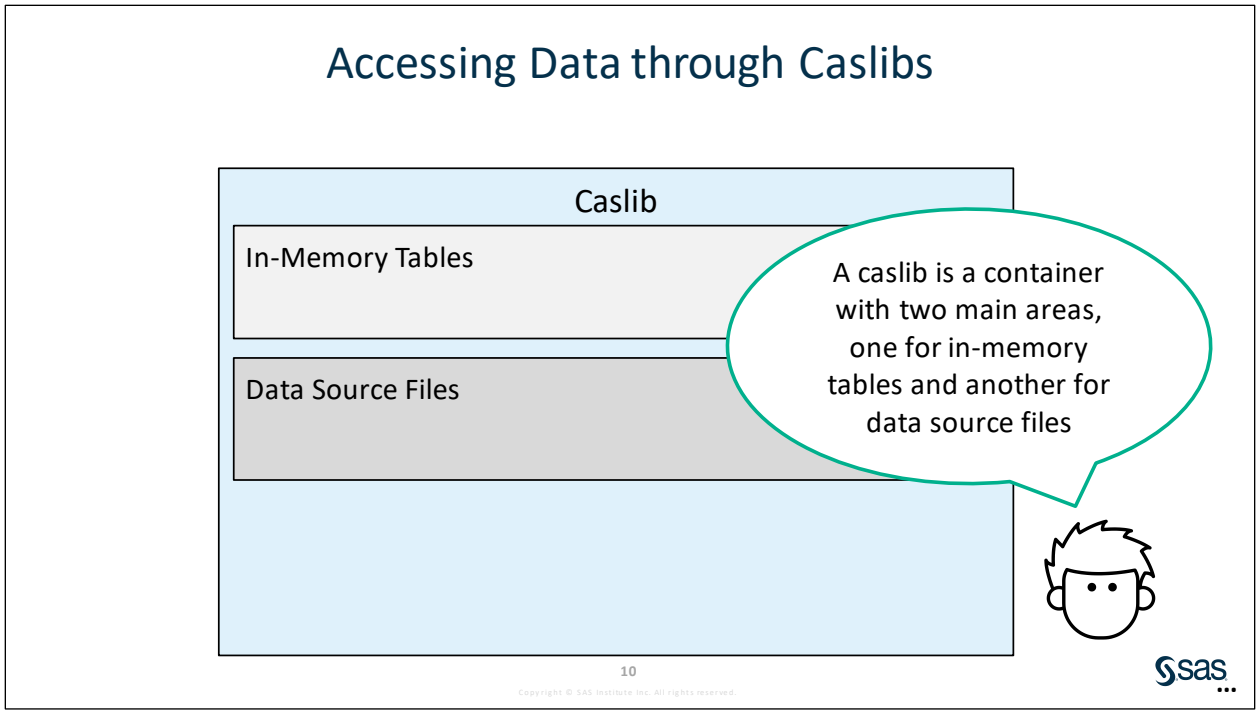
1. In the Explorer section of the navigation pane, expand ***<server>*** ⇨ **Home** ⇨ **Courses** ⇨ **PGVY35** ⇨ **activities**. Open the **pv02a01.sas** program.
2. Highlight **<insert path>** in the LIBNAME statement. In the Explorer, right-click the **PGVY35/data** folder and select **Insert as path**.
   Note: UNIX and Linux paths are case sensitive, so the **Insert as path** option is recommended to ensure correct casing in the text.
3. Run the code and verify that the library was successfully assigned.
4. In the Libraries section, expand the **pvbase** library.
5. Why are the Microsoft Excel and text files from the data folder not visible in the SAS library?

6

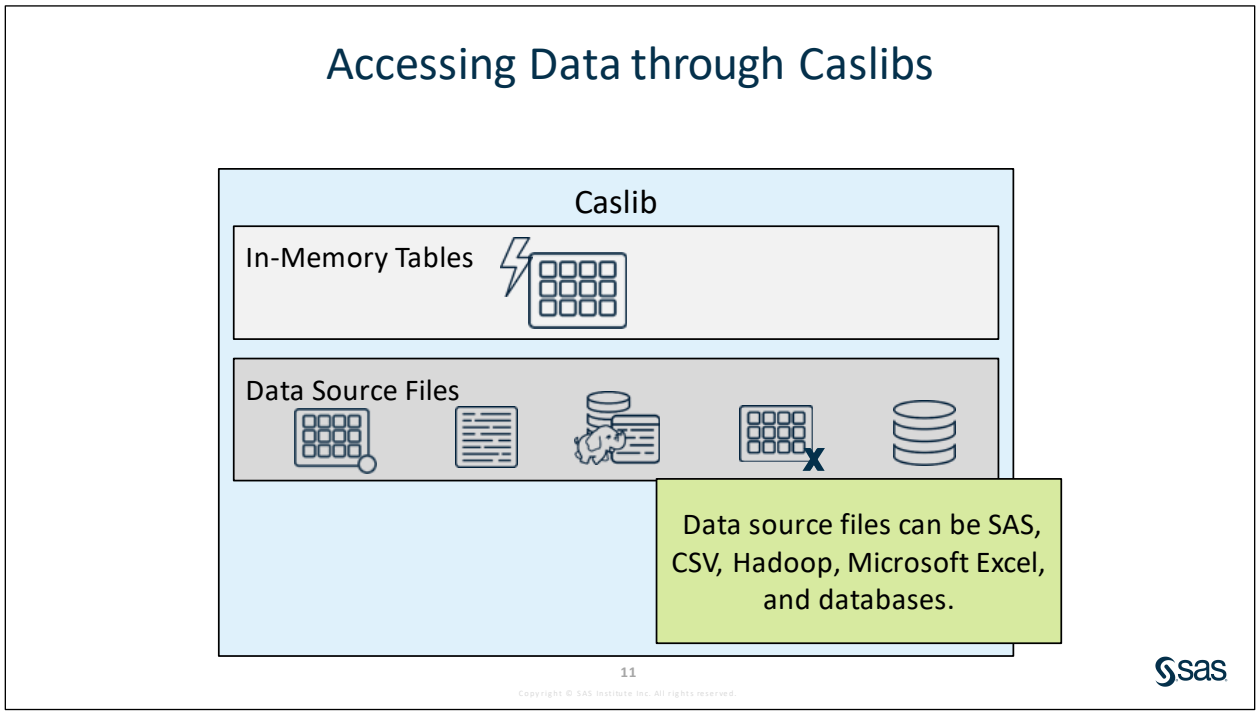
2.02 Activity

1. In the Explorer section of the navigation pane, right-click the **PGVY35** folder and select **Add as shortcut**.
2. Enter **PGVY35** in the **Name** field and click **OK**.
3. In the Explorer, expand **Folder Shortcuts** ⇨ **PGVY35** to access the files for this course.

8



Libraries in SAS Cloud Analytics Services are called *CAS libraries*, or *caslib*s. A caslib is the mechanism by which data is accessed in the CAS server. At its simplest, a caslib is a container that has two main areas: one for in-memory tables and another for data source files.



Data source files can include a variety of different types of data, including SAS, CSV, Hadoop, Microsoft Excel, and a variety of databases.

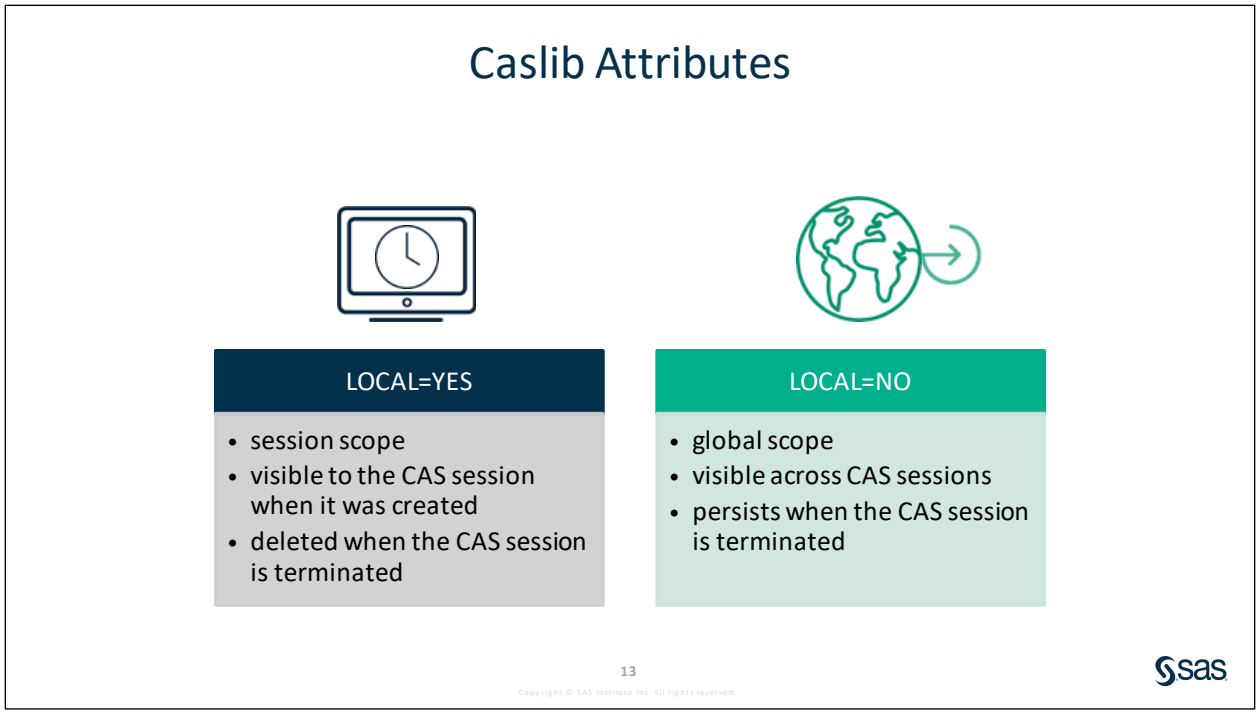
Caslib Attributes

Each caslib is defined by attributes that describe the data connection and user access.

Caslib Information	
Library	PVCAS
Source Type	PATH
Path	/home/student/Courses/PGVY35/data/
Session local	Yes
Active	Yes
Personal	No

12

A caslib is defined by several attributes that establish the connection to data source files and define user access. Let's look more closely at some of these key attributes.



The LOCAL attribute indicates the scope of the caslib. Caslibs can have either global scope or session scope.

When LOCAL=YES, the caslib is session scope. A session-scope caslib is available only to the CAS session where it was created. When the CAS session ends, the session-scope caslib is deleted. Session-scope caslibs are useful when you do not need to share data across sessions.

When LOCAL=NO, the caslib is global scope, which means that the caslib is available to anyone who has access permission. Global-scope caslibs are useful if you want to share data across sessions. When the CAS session ends, the global-scope caslib is ***not*** deleted. Because it is global, it persists between CAS sessions.

Caslib Attributes

ACTIVE=YES	ACTIVE=NO
• current default or active caslib • tables and files are assumed to be in the active caslib	• caslib is available • caslib name must be specified if referencing tables or files

14

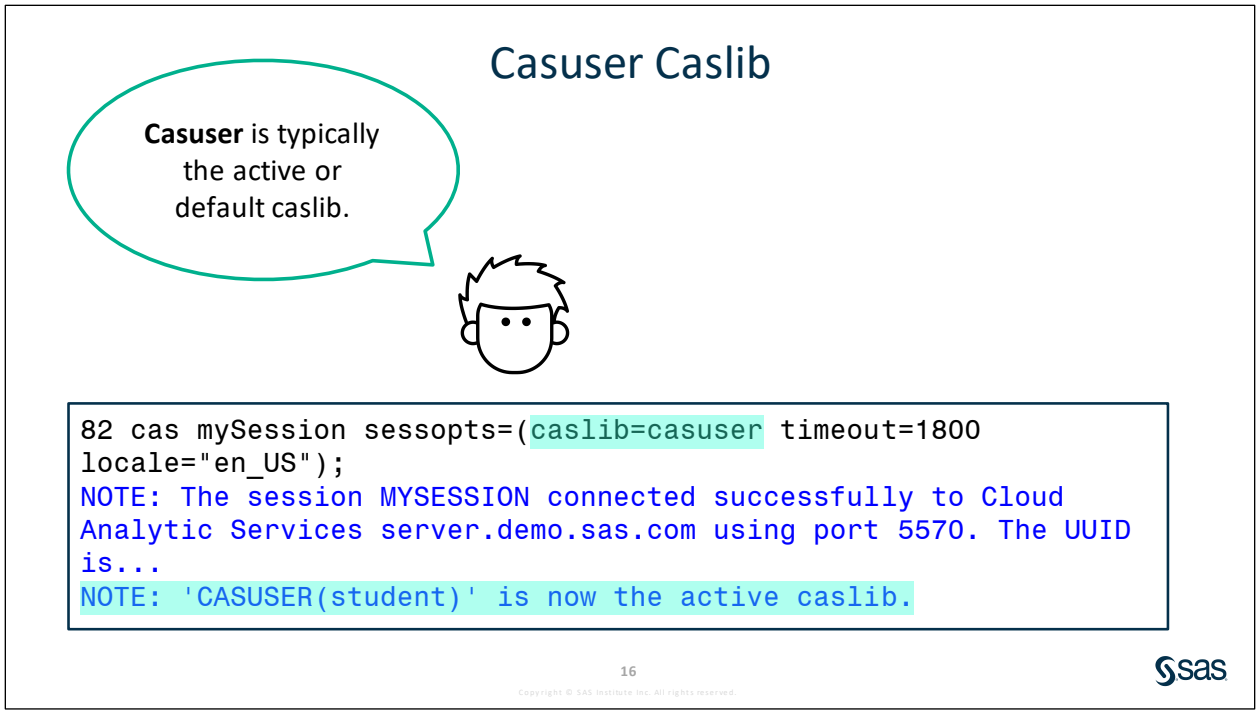
The ACTIVE attribute indicates the default caslib. The default caslib has ACTIVE equal to YES, which means that it is not required to specify the caslib in code when referencing a data source. Caslibs with ACTIVE=NO can also be used in code. However, the caslib must be specified along with the data source.

Caslib Attributes

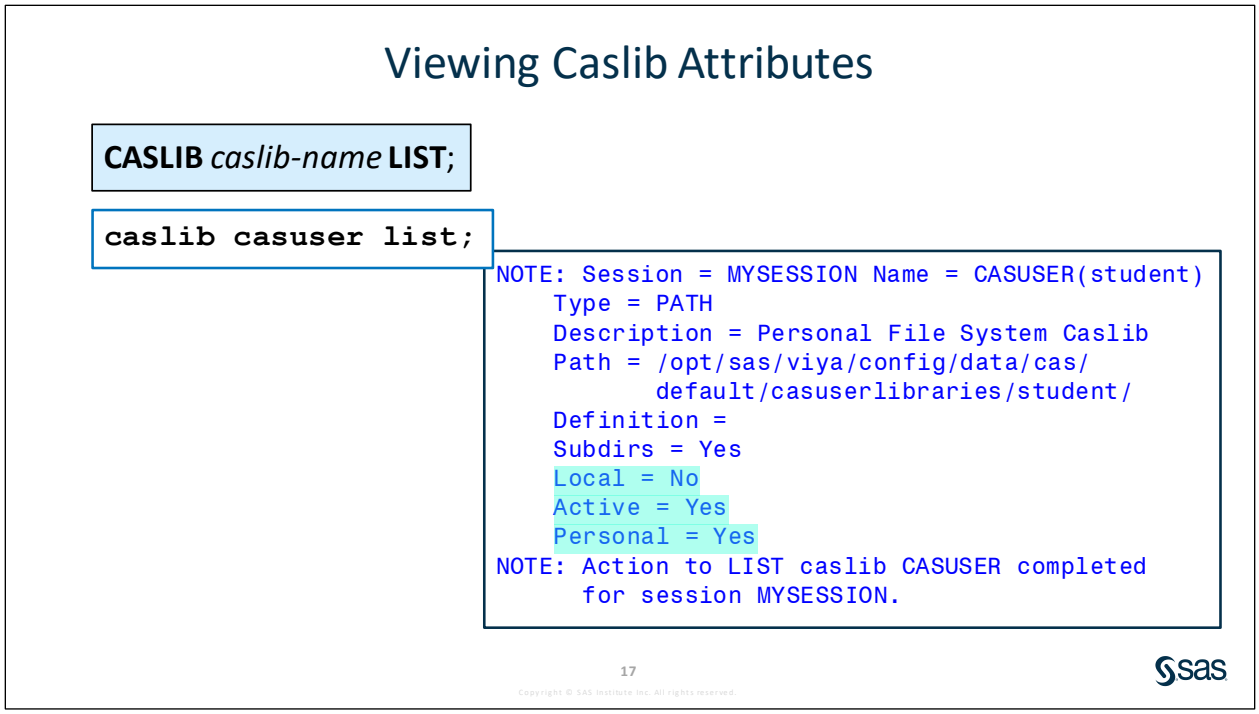
PERSONAL=YES	PERSONAL=NO
• global scope • only your user ID can access the data	• all other caslibs

15

When the PERSONAL attribute is YES for a caslib, it is available only to you. Think of it as your own personal place to work with data in CAS.

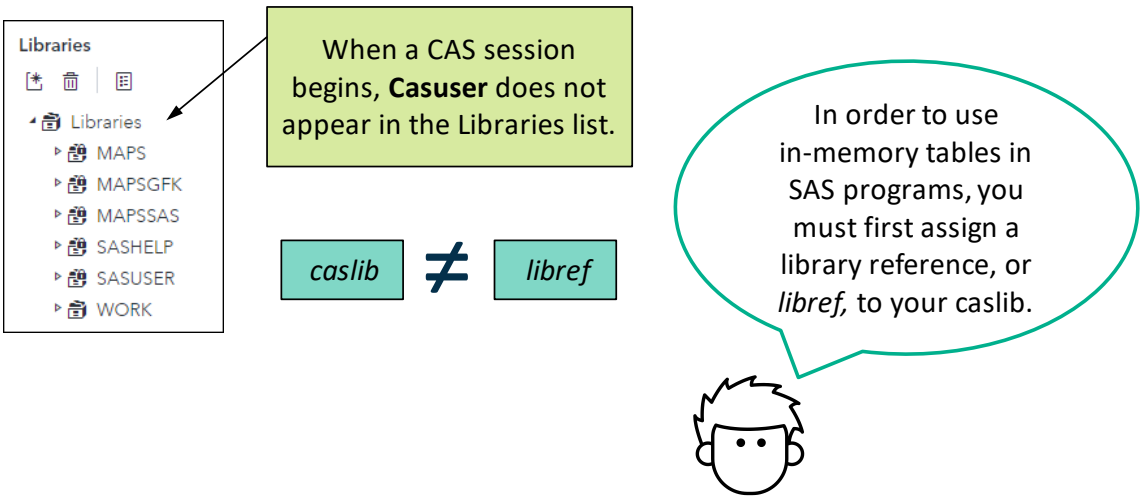


When you connect to a CAS session, **Casuser** is typically the active caslib. It is a personal, global-scope caslib. Data source files in **Casuser** are available only to you and can be used in any CAS session. A different active caslib can be specified in the CAS statement with the SESSOPTS and CASLIB= options.



To view the caslib attributes, you can submit the CASLIB statement with the LIST option. For example, here we submit **caslib casuser list;**. The information written to the log shows the session and caslib name, a description, the path, and other attribute values. LOCAL=NO means that **Casuser** is a global-scope caslib and will be available in each CAS session that you create. ACTIVE=YES means that it is the default caslib. PERSONAL=YES means that the caslib and the data sources included are available only to you.

Assigning a Library Reference to Caslibs



Although **Casuser** is automatically available as the default caslib when your CAS session begins, you cannot use it in place of a library reference, or *libref*, in a SAS program. To access in-memory tables in your code, you must first assign a libref to your caslib.

Assigning a Library Reference to Caslibs

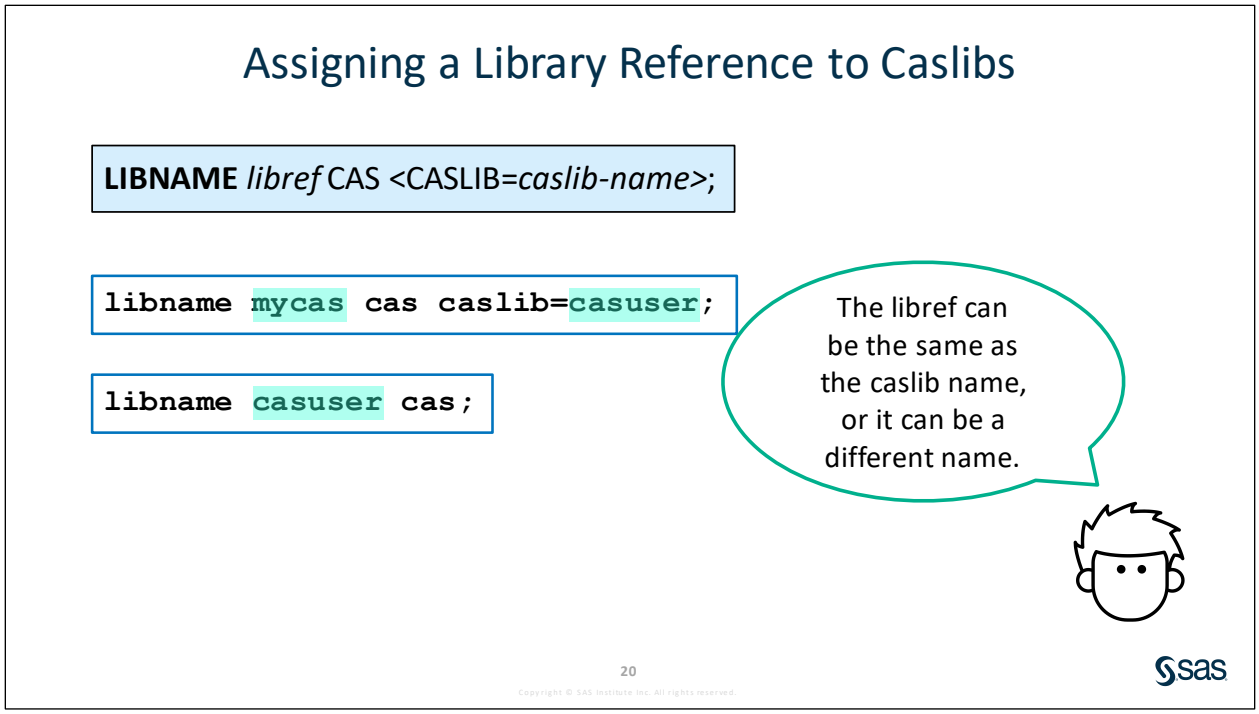
LIBNAME *libref* CAS <CASLIB=*caslib-name*>;

A library reference must be assigned to a caslib using the CAS engine.

19

sas

You can assign a library reference to use your caslib in two different ways. First, you can use the LIBNAME statement to create a library reference that points to your caslib. CAS is the required engine.



In the first LIBNAME statement example, the libref **mycas** points to the **casuser** caslib. In the second LIBNAME statement, the CASLIB= option is omitted. In that case, the libref **casuser** points to the caslib also named **casuser**. Note that the libref can be the same as or different from the caslib name.

Assigning a Library Reference to Caslibs

CASLIB _ALL_ ASSIGN;

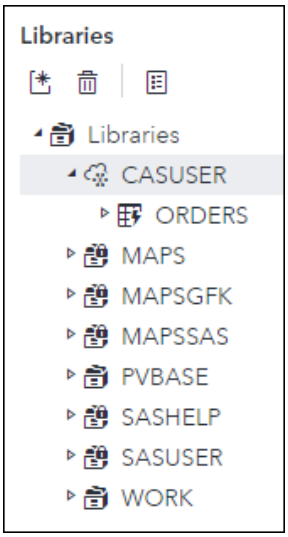
automatically assign a matching library reference to all existing caslibs

```
NOTE: A SAS Library associated with a caslib can only reference library
      member names that conform to SAS Library naming conventions.
NOTE: CASLIB CASUSER(student) for session MYSESSION will be mapped to
      SAS Library CASUSER.
NOTE: CASLIB PGVY35 for session MYSESSION will be mapped to SAS Library PGVY35.
NOTE: CASLIB Public for session MYSESSION will be mapped to SAS Library PUBLIC.
```

21

The second way to assign librefs is to use the CASLIB _ALL_ ASSIGN statement. This statement automatically assigns a library reference to each existing caslib in the session. It uses the caslib name as the library reference name. If the caslib name does not follow SAS naming rules, a libref is not assigned.

Assigning a Library Reference to Caslibs

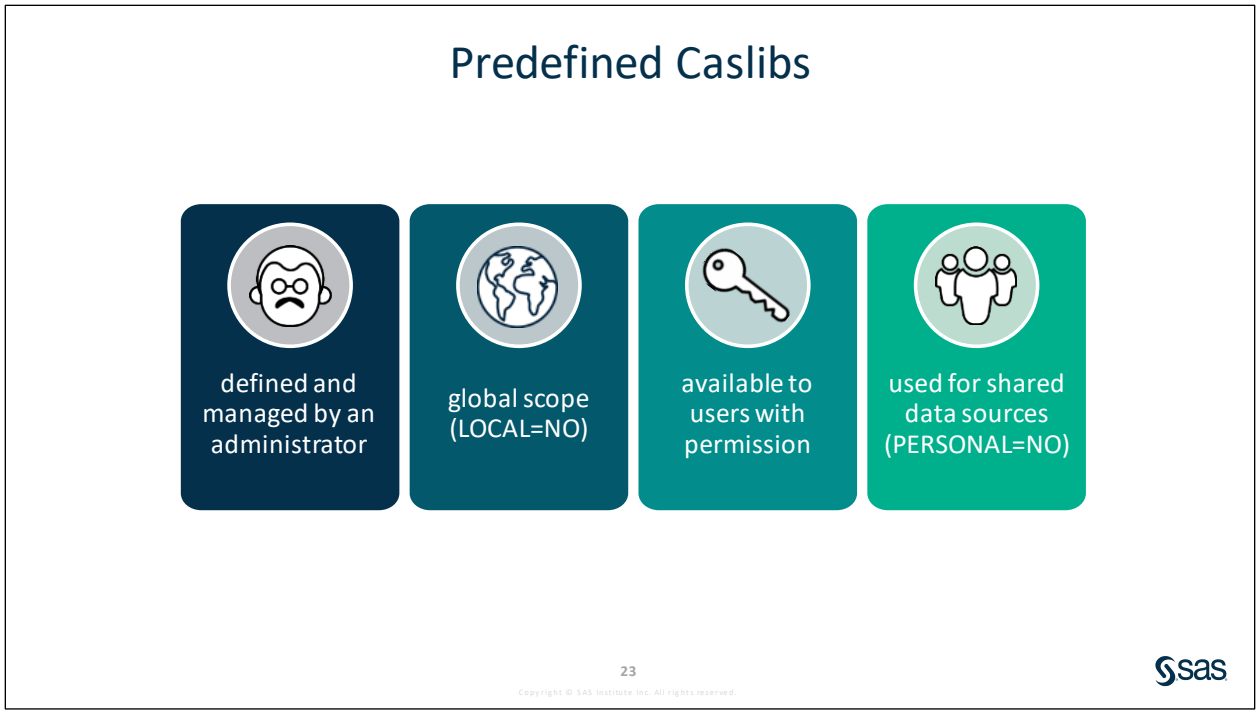


Librefs pointing to caslibs are visible in the Libraries panel and are available to use in SAS programs to reference in-memory tables.

```
proc means data=casuser.orders;
```

22

After librefs are assigned to caslibs, you can see them listed in the Libraries section with a cloud icon. You can also use standard syntax to reference in-memory tables such as *libref.table-name*.



Caslibs and corresponding librefs can be predefined for you by your administrator. Predefined caslibs have global scope, and the administrator sets data access controls. Predefined caslibs typically contain popular shared data sources for your site.

Accessing Caslibs

Scenario

Access and investigate the available caslibs in SAS Viya.

Program

- **pv02d01.sas**

Syntax

CASLIB _ALL_ LIST | _ALL_ ASSIGN**;**

LIBNAME *libref* CAS <*options*>**;**

Notes

- When used with the _ALL_ keyword and the LIST option, the CASLIB statement displays all the caslibs that the user can access in his or her session and the caslib specifications.
- When used with the _ALL_ keyword and the ASSIGN option, the CASLIB statement assigns SAS library references.
- The CASLIB statement with the ASSIGN option is used with the _ALL_ keyword to assign SAS library references for existing caslibs so that they are visible in the SAS Studio libraries tree.
- Individual caslibs can be made visible in the SAS Studio libraries tree with a CAS LIBNAME statement. The CAS LIBNAME statement associates a SAS libref with in-memory tables on the CAS server.

Demo

1. Begin with a new SAS session by selecting **Options** ⇨ **Reset SAS session**. Click **Reset** when asked to confirm.
2. In the Explorer section of the navigation pane, expand the **PGVY35** folder shortcut. In the **demos** folder, open the **pv02d01.sas** program.
3. Highlight **<insert path to data folder>**. Right-click the **data** folder in the Explorer section and select **Insert as path**.
4. Highlight and submit the first three statements to start the CAS session and list the available caslibs in the session.

   Note: Your path to the **data** folder might be different.

```
libname pvbase base "/home/student/Courses/PGVY35/data";

cas mySession sessopts=
    (caslib=casuser timeout=1800 locale="en_US");

caslib _all_ list;
```

5. Examine the log and verify the following:
   a. The **pvbase** library was successfully assigned.
   b. The CAS session **mySession** is connected and **Casuser** is the active caslib. The SAS log notes will be slightly different if you did not start a CAS session in a previous activity.

      Note: The active caslib is the default caslib in the CAS session.

   c. A list of caslibs that are available in the CAS session and their attributes are displayed in the log.

```
82   libname pvbase base "/home/student/Courses/PGVY35/data";
NOTE: Libref PVBASE was successfully assigned as follows:
      Engine:        BASE
      Physical Name: /home/student/Courses/PGVY35/data
83   cas mySession sessopts=(caslib=casuser timeout=1800 locale="en_US");
NOTE: The session MYSESSION connected successfully to Cloud Analytic Services
server.demo.sas.com using port 5570. The UUID is
      d6117b19-b88e-7f4a-bd0a-1f34ed963c79. The user is student and the active caslib is
CASUSER(student).
NOTE: The SAS option SESSREF was updated with the value MYSESSION.
NOTE: The SAS macro _SESSREF_ was updated with the value MYSESSION.
NOTE: The session is using 0 workers.
NOTE: 'CASUSER(student)' is now the active caslib.
NOTE: The CAS statement request to update one or more session options for session MYSESSION
completed.
84   caslib _all_ list;
NOTE: Session = MYSESSION Name = CASUSER(student)
         Type = PATH
         Description = Personal File System Caslib
         Path = /opt/sas/viya/config/data/cas/default/casuserlibraries/student/
         Definition =
         Subdirs = Yes
         Local = No
         Active = Yes
         Personal = Yes
...
```

6. In the Libraries section of the navigation pane, confirm that the **pvbase** library appears in the SAS libraries tree. However, you do not see any of the caslibs. By default, the caslibs do not show up in the SAS libraries tree until a libref has been assigned with the CAS engine.
7. Go back to the **Code** tab. Highlight and submit the following statement:

```
libname casuser cas caslib=casuser;
```

   This statement creates a CAS engine library reference for the in-memory tables in the **Casuser** caslib:

```
NOTE: Libref CASUSER was successfully assigned as follows:
      Engine:        CAS
      Physical Name: 26297f9f-a9e3-f242-ac58-c0ded098cb46
```

8. In the Libraries section, confirm that a SAS library reference for the **Casuser** caslib is now visible. However, there are still no tables listed. This is because **Casuser** does not currently have any tables loaded into memory.

9. Go back to the **Code** tab. Highlight and submit the following statement:

```
caslib _all_ assign;
```

   The ASSIGN option, when used with the _ALL_ keyword, creates SAS library references for existing caslibs so that they are visible in the SAS Studio libraries tree. The library reference names will match the caslib names.

10. Check the log to see which caslibs were mapped to SAS libraries.

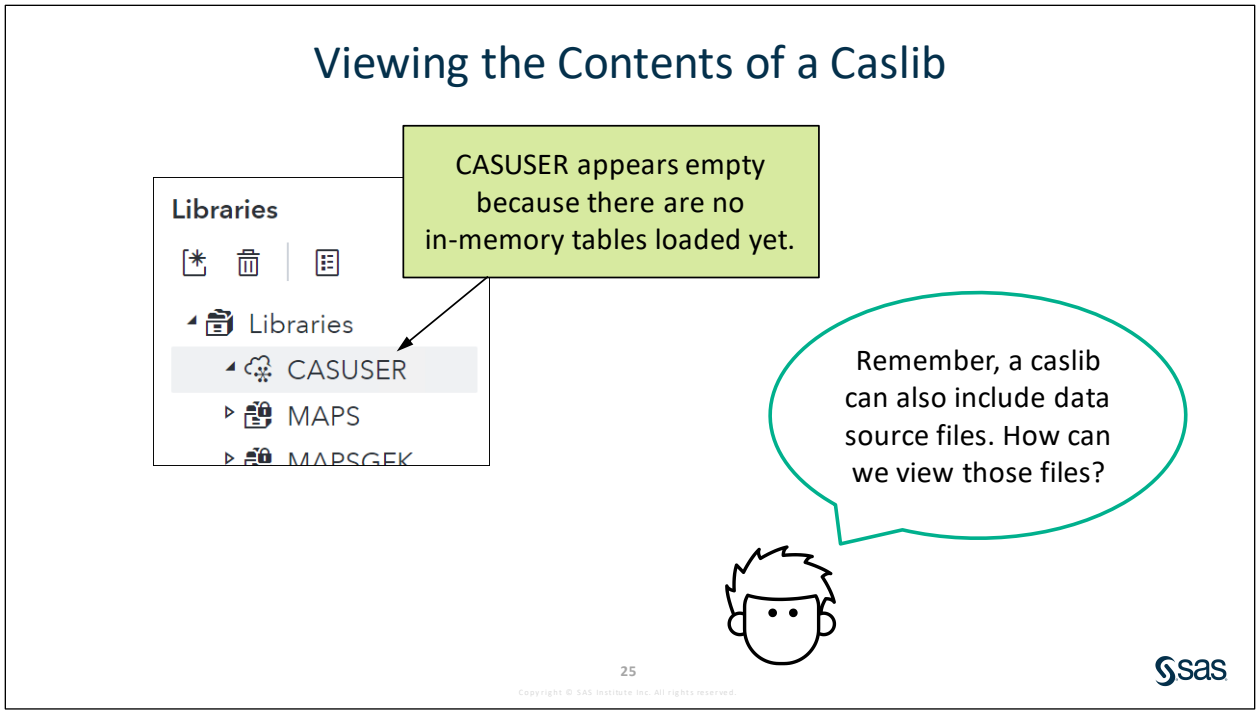
    Note: The libraries listed might differ, depending on your environment.

```
82   caslib _all_ assign;
NOTE: A SAS Library associated with a caslib can only reference library member names that
conform to SAS Library naming conventions.
NOTE: CASLIB CASUSER(student) for session MYSESSION will be mapped to SAS Library CASUSER.
NOTE: CASLIB DIDP for session MYSESSION will be mapped to SAS Library DIDP.
NOTE: CASLIB Formats for session MYSESSION will be mapped to SAS Library FORMATS.
NOTE: CASLIB ModelPerformanceData for session MYSESSION will not be mapped to SAS Library
ModelPerformanceData. The CASLIB name is not valid for use as a libref.
NOTE: CASLIB Models for session MYSESSION will be mapped to SAS Library MODELS.
NOTE: CASLIB Public for session MYSESSION will be mapped to SAS Library PUBLIC.
NOTE: CASLIB Samples for session MYSESSION will be mapped to SAS Library SAMPLES.
NOTE: CASLIB SystemData for session MYSESSION will not be mapped to SAS Library SystemData.
The CASLIB name is not valid for use as
      a libref.
```

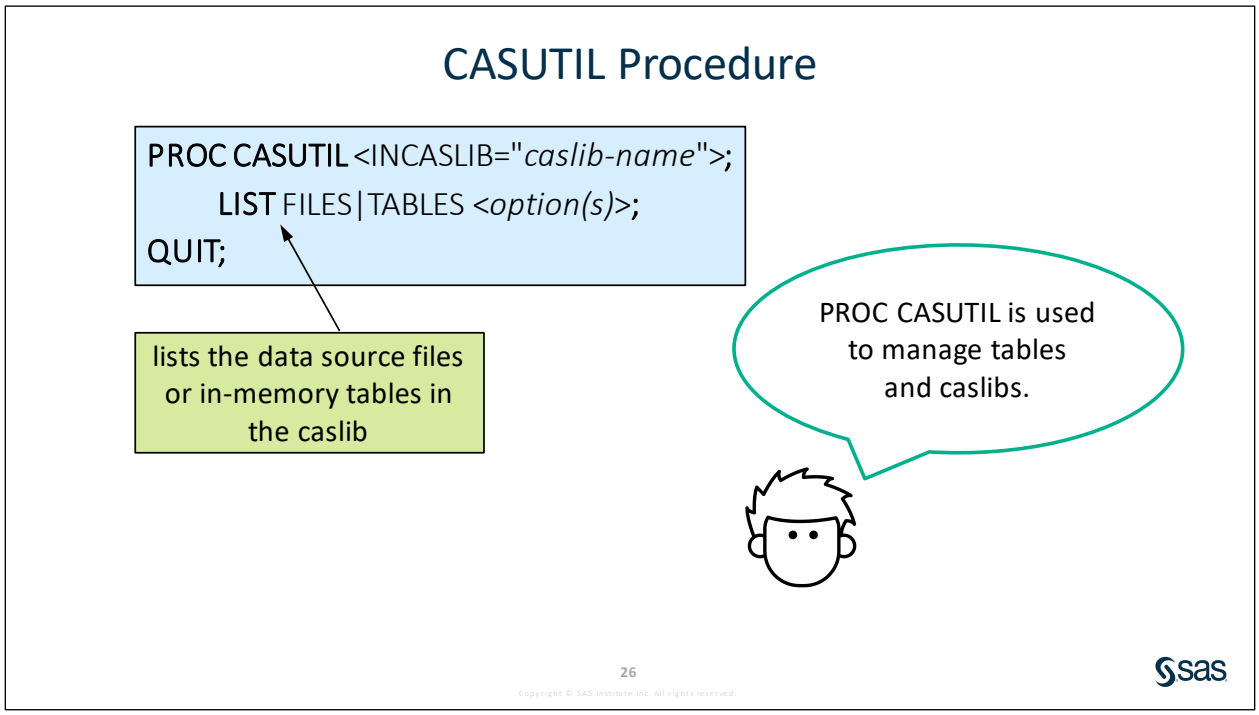
11. In the Libraries section, confirm that the SAS library references for multiple caslibs are now visible in the SAS libraries tree.

    Note: Caslibs with names longer than eight characters were not added to the tree, as indicated in the SAS log notes.

End of Demonstration



After a libref is assigned to a caslib using the CAS engine, the caslib is accessible in the Libraries section of the navigation pane. However, only in-memory tables are displayed. At this point, we have data source files in **Casuser**, but we have not loaded anything to memory. So how can we view data source files that are available in a caslib?



The CASUTIL procedure is a helpful procedure to manage caslibs, data source files, and in-memory tables. If the INCASLIB= option is not specified, the active caslib is the default. The LIST statement lists either the data source files or in-memory tables.

2.03 Activity

Open **pv02a03.sas** from the **activities** folder and perform the following tasks:

1. Add the LIST FILES statement to the PROC CASUTIL step. Run the program.

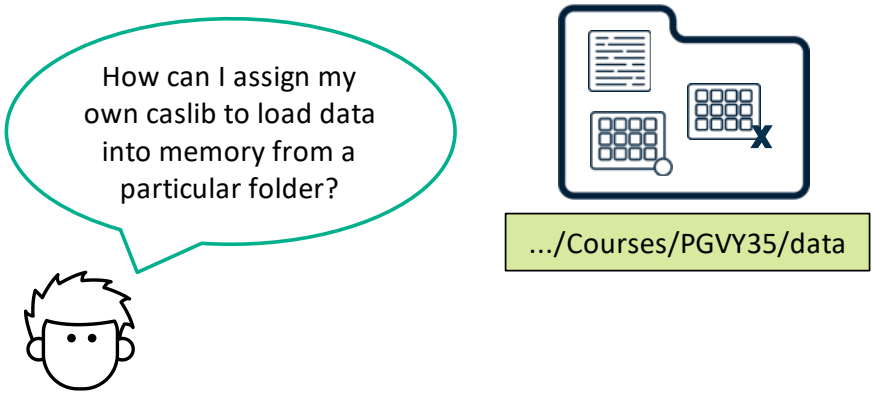
```
list files;
```

2. Which caslib is analyzed in the report? How many data source files are in the caslib?
3. Change the FILES option to TABLES. Run the program and view the log. How many in-memory tables are in the caslib?

```
list tables;
```

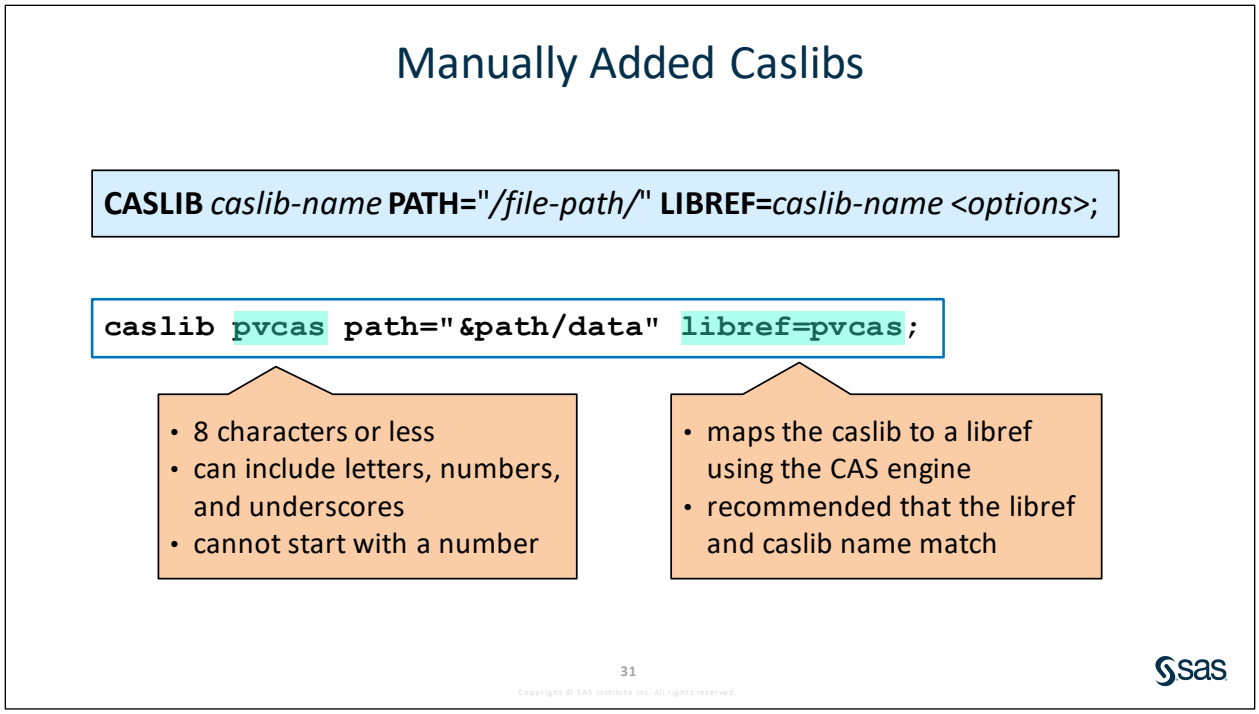
27

Manually Added Caslibs



30

Let's say that you have several data source files in a particular folder, and you would like to load data files from that location into memory. If you are authorized by your administrator, you can manually add your own caslibs to access and process that data in CAS.



To manually add a caslib to reference data source files in a folder, you use the CASLIB statement. You provide a name of your choice for the caslib. Keep in mind that the caslib name must be eight characters or less and follow standard SAS naming conventions. The PATH= option specifies the location of the caslib. The LIBREF= option maps the caslib to a library reference, using the CAS engine. The libref can also be a name of your choice, but it is recommended that the libref match the caslib name.

This example is specifically for assigning a caslib for a path, but additional options are available to add caslibs to point to other data sources.

Manually Added Caslibs

```
89 caslib pvcas path="&path/data" libref=pvcas;
NOTE: 'PVCAS' is now the active caslib.
NOTE: Cloud Analytic Services added the caslib 'PVCAS'.
NOTE: CASLIB PVCAS for session MYSESSION will be
      mapped to SAS Library PVCAS.
NOTE: Action to ADD caslib PVCAS
      completed for session MYSESSION.
```

When you define a new caslib, it becomes the active caslib for the session.

32

When a new caslib is defined, it automatically becomes the new active caslib. By default, manually added caslibs are session scope, but they can be created with global scope.

Manually Added Caslibs

```
proc casutil;
    list files;
quit;
```

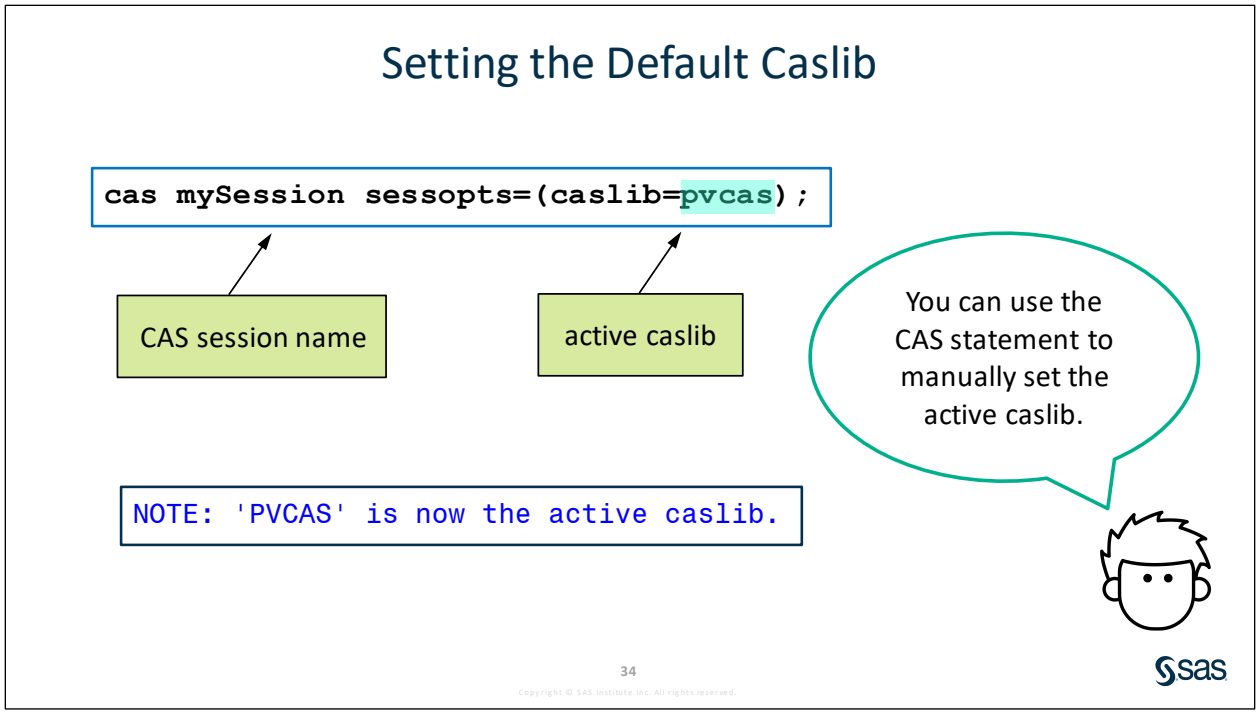
Caslib Information	
Library	PVCAS
Source Type	PATH
Path	/home/student/Courses/PGVY35/data/

Data source files in the folder are now accessible through the **Pvcas** caslib.

CAS File Information					
Name	Permission	Owner	Group	Encryption Method	File Size
country_lookup.sas7bdat	-rwxr--r--	student	sasusers		72.0KB
customers.xlsx	-rwxr--r--	student	sasusers		4.1MB
employees.sas7bdat	-rwxr--r--	student	sasusers		256.0KB
orders.csv	-rwxr--r--	student	sasusers		245.2MB
...					

33

The new caslib now provides access to the data source files that are located in the path. These files are not yet loaded into memory, but that will be our next step.



You can change the active caslib with the CASLIB= session option in the CAS statement. For example, the following CAS statement sets the active caslib to **Pvcas**.

Defining a New Caslib

Scenario

Create a new caslib to access data source files in a path and investigate the attributes of the caslib.

Program

- **New CAS Session** snippet
- **New caslib for Path** snippet

Syntax

CASLIB *caslib-name* **PATH="**/*file-path*/**" LIBREF=***caslib-name* <*options*>**;**

CASLIB *caslib-name* **CLEAR;**

Notes

- Use the CASLIB statement to define a caslib.
- The PATH= option defines the location of the data source files.
- A library reference must be mapped to the caslib. This can be accomplished with either the LIBREF= option in the CASLIB statement or a LIBNAME statement that uses the CAS engine. It is recommended to match the library reference with the caslib name.
- The CASLIB statement with the CLEAR option deletes a caslib.

Demo

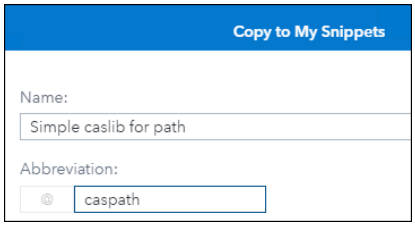
Note: If you do not have an active CAS session, first submit the **New CAS Session** snippet.

1. In the Snippets section, double-click the **New caslib for Path** snippet to open the program. This snippet is a good starting point for creating a new caslib, but it can be simplified. Make the following modifications:
   a. Delete the DATASOURCE=, SESSREF=, and SUBDIRS options.
   b. Add the LIBREF= option.
   c. Delete the LIBNAME statement.
   d. Delete lines 3 through 5 in the comment block.

   Note: The DATASOURCE= and SESSREF= options are included in the snippet but are not required when creating a caslib based on a path in the current CAS session. The LIBNAME statement is not needed if the LIBREF= option is used in the CASLIB statement.

```
/*************************************************************/
/*  Create a CAS library (myCaslib) for the specified path
    ("/filePath/")                                           */
/*************************************************************/
caslib myCaslib path="/filePath/" libref=myCaslib;
```

2. Select **Copy to My Snippets**. Enter **Simple caslib for path** in the **Name** field and **caspath** in the **Abbreviation** field. Click **OK**.



3. Create a new program tab. Add the CAS statement to create a new CAS session named **mysession** if it is not already active.

   Note: If the SESSOPTS option is not specified, default values are assigned for all session attributes.

```
cas mysession;
```

4. On the next line in the program, type **@caspath** and press the Enter key. The snippet code is inserted into the program. Make the following modifications:

   Note: You can also expand **My Snippets**, then right-click **Simple caslib for path** and select **Insert**.

   a. Change the name of the caslib to **pvcas** after CASLIB and LIBREF=.

   b. Highlight the text within the quotation marks in the PATH= option. In the Explorer section, right-click the **PGVY35/data** folder and select **Insert as path**.

      Note: Your path might be different, depending on your environment.

```
caslib pvcas path="/home/student/Courses/PGVY35/data"
              libref=pvcas;
```

5. Run the program and view the log. Confirm that the **pvcas** caslib was added and that it is now the active caslib. Also confirm that **pvcas** appears in the Libraries section.

6. Add a PROC CASUTIL step to list the files in the active caslib. Submit the step and confirm that several Excel, CSV, and SAS7BDAT files are included. Also, notice in the Caslib Information section that the Session local value is Yes, which means that the caslib is session scope.

```
proc casutil;
    list files;
quit;
```

The CASUTIL Procedure

Caslib Information	
Library	PVCAS
Source Type	PATH
Path	/home/student/Courses/PGVY35/data/
Session local	Yes
Active	Yes
Personal	No
Hidden	No
Transient	No

7. Select **Options** ⇨ **Reset SAS session** ⇨ **Reset**. Notice that **pvcas** no longer appears in the Libraries section. Because the **pvcas** caslib is session scope (LOCAL=YES), it is cleared when the SAS session ends.
8. To clear the caslib without terminating the CAS session, add the following CASLIB statement at the end of the program:

```
caslib pvcas clear;
```

9. Run the entire program and view the log. The CAS session starts and **Casuser** is the active caslib. Next, the **pvcas** caslib is assigned and automatically becomes the active caslib. After the **pvcas** caslib is cleared, **Casuser** becomes the active caslib.

```
NOTE: 'CASUSER(student)' is now the active caslib.
NOTE: Cloud Analytic Services removed the caslib 'PVCAS'.
NOTE: Action to DROP caslib PVCAS completed for session MYSESSION.
```

End of Demonstration

2.04 Activity

Open **pv02a04.sas** from the **activities** folder and perform the following tasks:

1. Modify the first CASLIB statement to create a caslib named **Pvcas** that points to the **/PGVY35/data** folder.

```
caslib pvcas path="<insertpath>" libref=pvcas;
```

2. Modify the last CASLIB statement to clear the **Pvcas** caslib after the PROC CASUTIL step.
3. Run the program and view the Caslib Information report. Is the Personal attribute Yes or No?

36

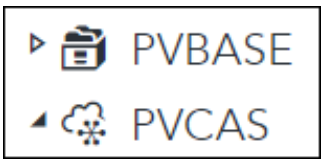
sas

2.05 Activity

Open **pv02a05.sas** from the **activities** folder and perform the following tasks:

1. Insert the path to the **data** folder in the %LET statement to create a macro variable named **Path**.
2. Run the program. The **Pvbase** and **Pvcas** librefs both point to the **PGVY35/data** folder. How are they different?

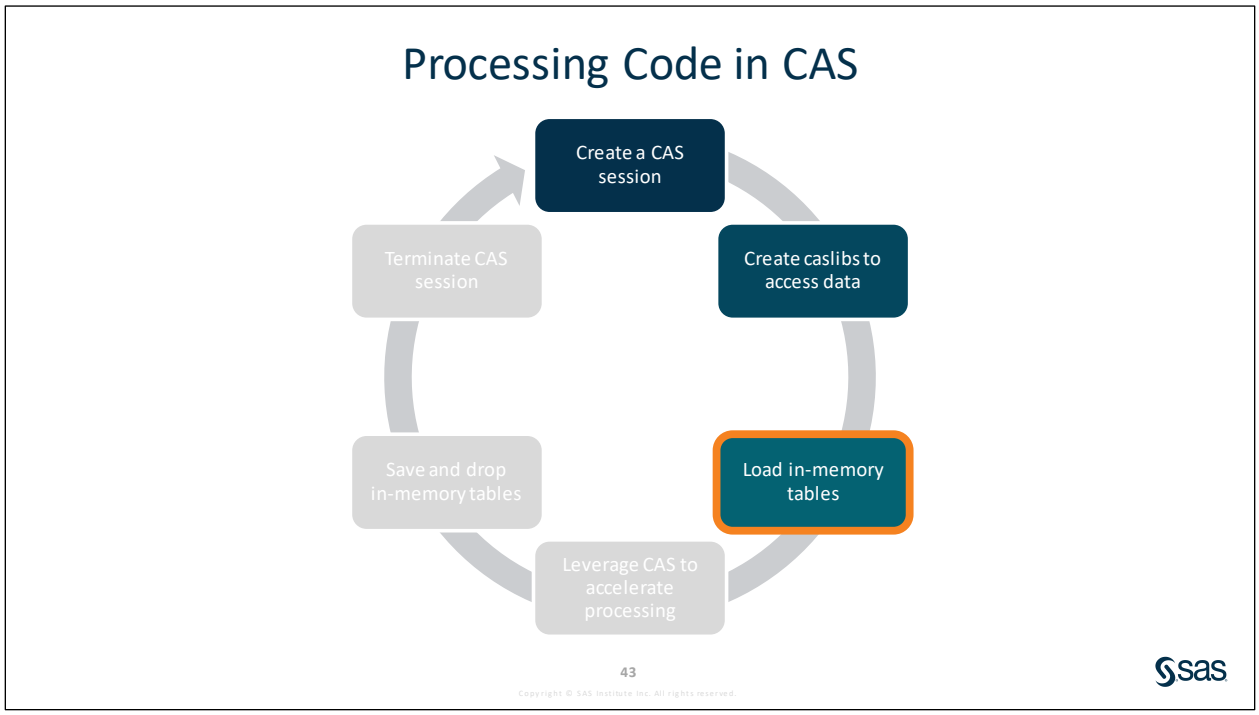
```
libname pvbase "&path";
caslib pvcas path="&path" libref=pvcas;

proc contents data=pvbase._all_ nods;
run;

proc contents data=pvcas._all_ nods;
run;
```



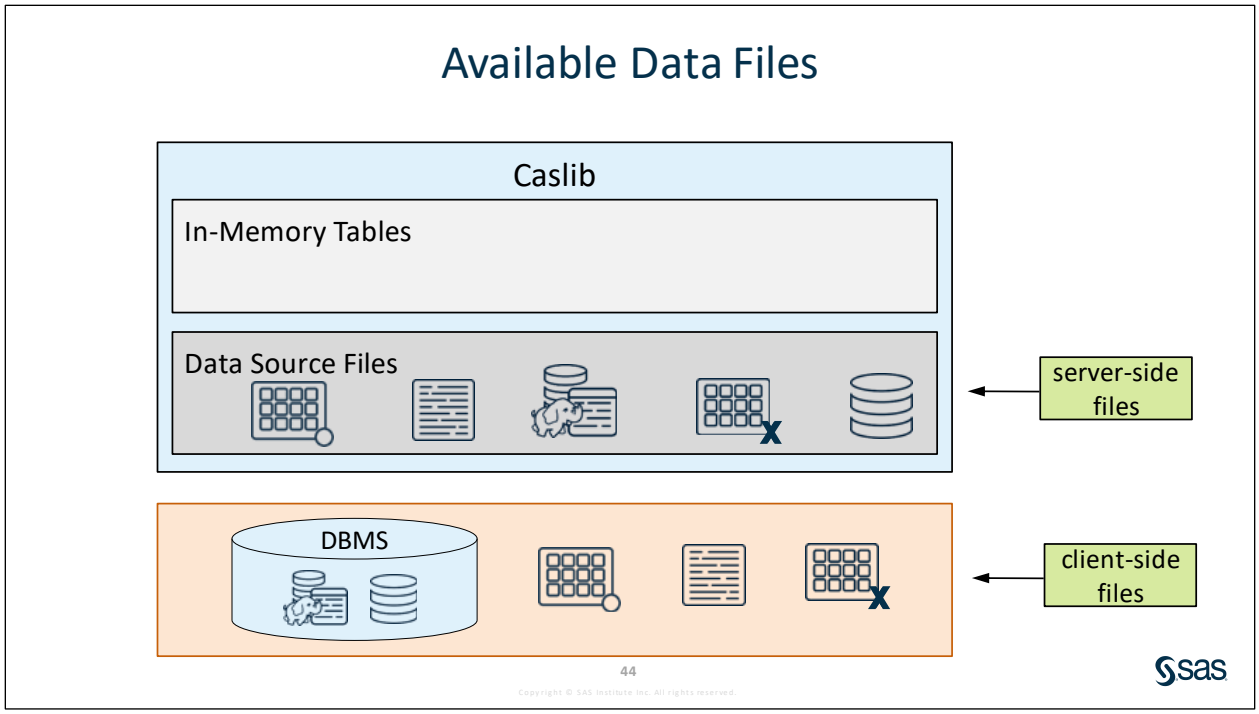
38

sas

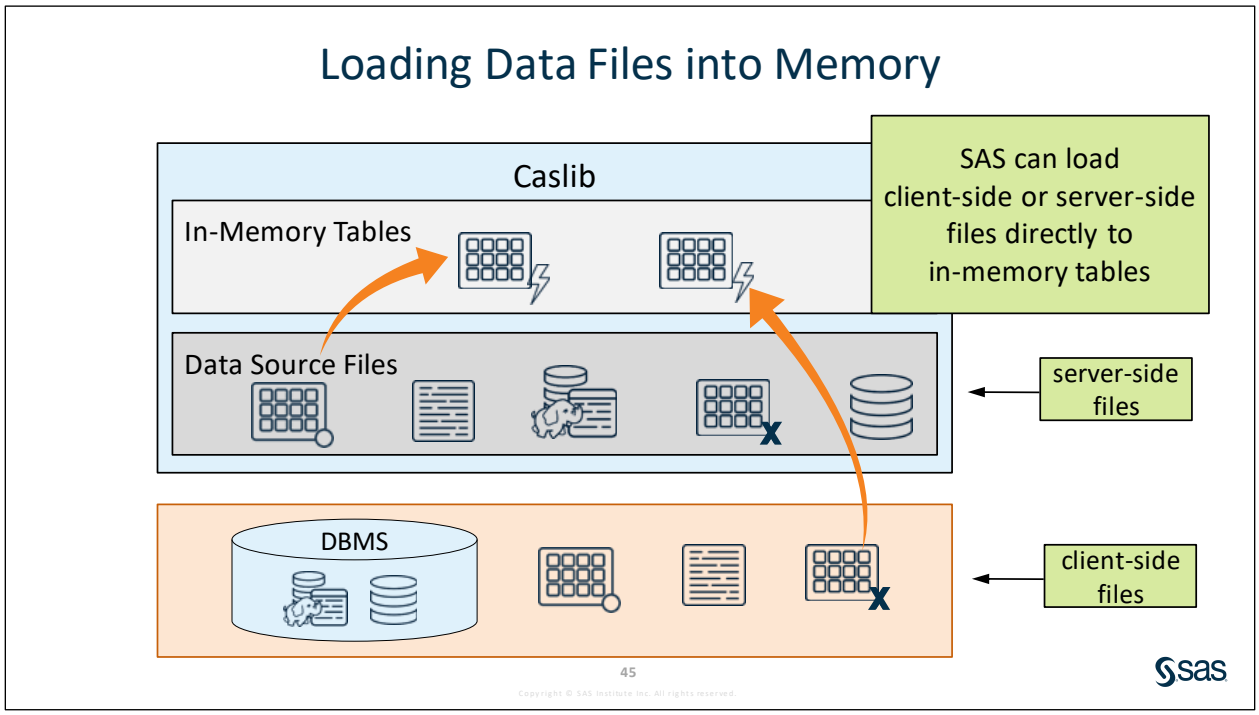
2.2 Loading Data to In-Memory Tables



You have started a CAS session and accessed a caslib. Now let's update or analyze data in a file. CAS can process data only in its in-memory space, so your next step is to load a data source file into memory.

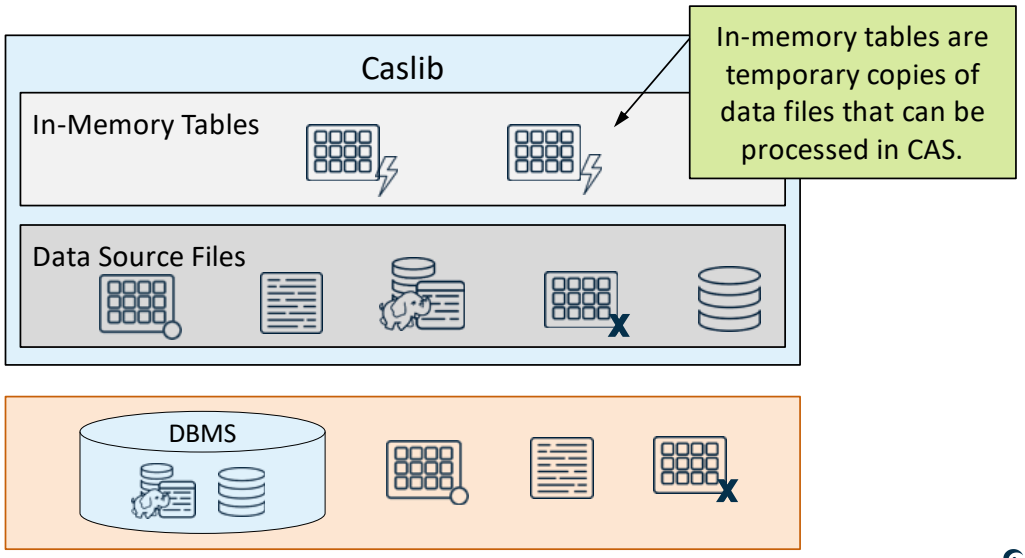


There are two sources of data files that can be loaded to in-memory tables in a caslib. Data files that are already included in a defined caslib are referred to as *server-side files*. Files of any kind that are not mapped to the caslib are called *client-side files*. Any data that you can access through the SAS Compute Server is considered a client-side file, including database tables, SAS tables, or text files.



SAS can load both client-side and server-side files directly to in-memory tables.

Loading Data Files into Memory

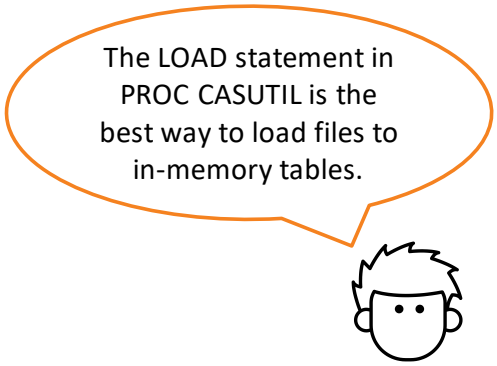


46

After a data source file is loaded into memory, it is referred to as an *in-memory table*. These in-memory tables are temporary ***copies*** of the associated data source file that can be processed in CAS. The data source files remain on disk and unchanged.

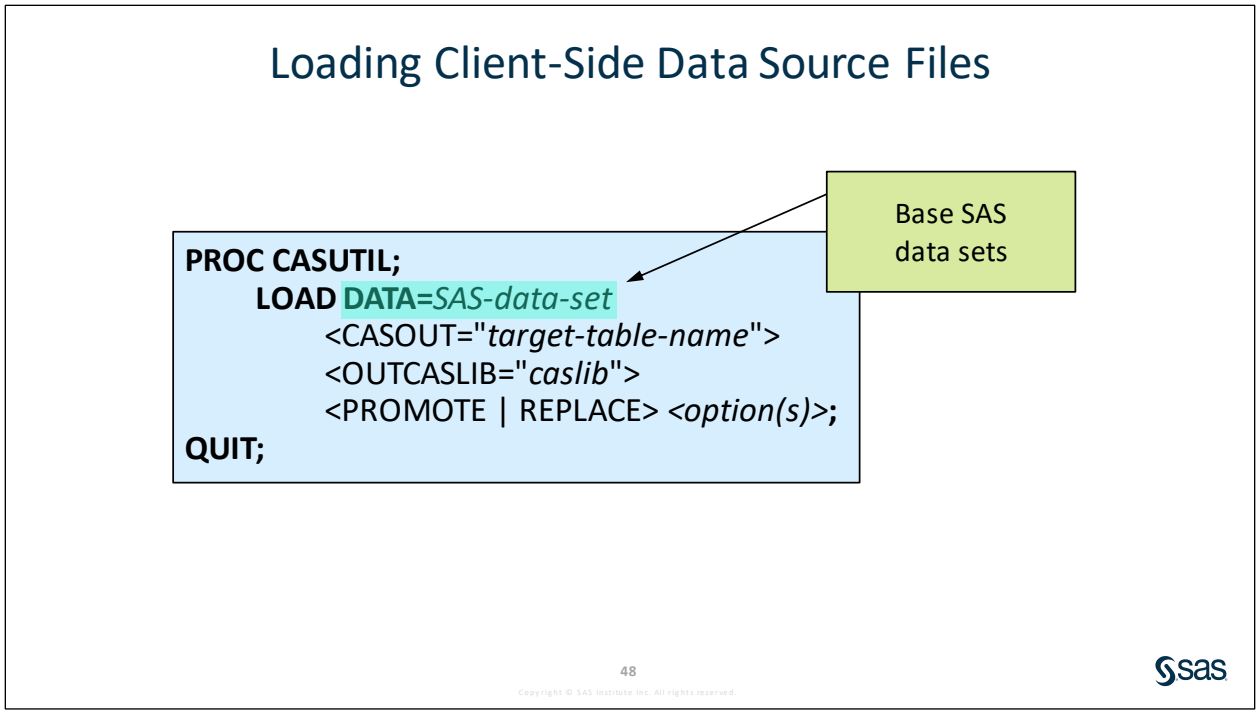
Loading Client-Side Data Source Files

```
PROC CASUTIL;
     LOAD <option(s)>;
QUIT;
```



47

The LOAD statement in PROC CASUTIL is typically the easiest and most efficient way to load files to in-memory tables.



The DATA= option in the LOAD statement is used for Base SAS data sets, such as **sashelp.cars** or **pvbase.orders**. CASOUT= names the in-memory table. OUTCASLIB= names the caslib in which the table will be loaded. PROMOTE creates a global-scope table. REPLACE overwrites data if the in-memory table already exists.

Notice that most option values are enclosed in quotation marks. The only exception is the SAS data set name after the DATA= option.

Loading Client-Side SAS Data into CAS

Scenario

Load a client-side SAS data set to an in-memory table in **Casuser** and investigate the descriptor portion of the table.

Programs

- **startup.sas**
- **Load Data to caslib** snippet

Syntax

```
PROC CASUTIL;
      LOAD DATA=SAS-data-set
            <OUTCASLIB="caslib">
            <CASOUT="target-table-name"> <PROMOTE | REPLACE> <option(s)>;
      LIST FILES <option(s)>;
      LIST TABLES <option(s)>;
      CONTENTS CASDATA="table-name" <INCASLIB="caslib"> <option(s)>;
      DROPTABLE CASDATA="table-name" <INCASLIB="caslib"><QUIET>;
QUIT;
```

Notes

- The INCASLIB= option is not necessary when loading a SAS data set into CAS. If you use the option, it is ignored.
- If you do not specify the OUTCASLIB= option, the table is copied into the active caslib.
- The CASOUT= option names the in-memory table. If that table name already exists in the caslib, use the REPLACE option to overwrite it.
- The LIST statement with the TABLES option lists the in-memory tables in a caslib.

Demo

1. **REQUIRED:** All subsequent demos and practices rely on having the **pvbase** SAS library defined, a CAS session started, and all caslibs assigned library references. The code to complete these actions is saved in the **startup.sas** program in your course files.

   a. Open the **startup.sas** program in the **PGVY35** folder. Highlight all the code and copy it (Ctrl+C) to the clipboard.

   b. In SAS Studio, click **Options** ⇨ **Autoexec File**.

   c. Paste (Ctrl+V) the program code into the box. Click **Run** and check the log to ensure that the program runs without errors.

   d. Click **Save**. Close the **startup.sas** program.

   Note: The **Autoexec.sas** program executes automatically every time that you sign in to SAS Studio. This ensures that the SAS Compute Server and CAS sessions are initialized for the classroom environment.

2. In the Snippets section of the navigation pane, expand **SAS Viya Cloud Analytics Services** and double-click **Load Data to caslib** to open the program.
3. This **Load Data to caslib** snippet includes sample code for loading client-side and server-side files to in-memory tables. The second PROC CASUTIL step contains a template to load a client-side SAS table to an in-memory table. Highlight the second comment block and PROC CASUTIL step, and then click **Copy to My Snippets**.
4. Make the following changes in the Copy to My Snippets window:
   a. Enter **Load client SAS table to caslib** in the **Name** field.
   b. Enter **loadsas** in the **Abbreviation** field.
   c. In the Insert code panel, change the RUN statement to a QUIT statement.
   d. Click **OK**.
5. Open a new program tab. Enter **@loadsas** and then press the Enter key. The custom snippet is inserted into the program.
6. Modify the LOAD statement to read **pvbase.employees** and load it to an in-memory table named **emps_cas** in the **Casuser** caslib. Add the REPLACE option to replace **emps_cas** if it is already in memory.

   Note: If the OUTCASLIB= option is not included, the table will be loaded to the default caslib.

```
proc casutil;
    load data=pvbase.employees outcaslib="casuser"
         casout="emps_cas";
quit;
```

7. Run the program and confirm in the log that **pvbase.employees** was successfully added to the **Casuser** caslib as **EMPS_CAS**.

```
87   proc casutil;
NOTE: The UUID 'a6590aa6-42eb-6e4e-8073-8545b7889d96' is connected using session MYSESSION.
88
88 !  load data=pvbase.employees outcaslib="casuser"
89   casout="emps_cas";
NOTE: PVBASE.EMPLOYEES was successfully added to the "CASUSER(student)" caslib as "EMPS_CAS".
90   quit;
```

8. In the Libraries section, expand **Libraries** ⇨ **CASUSER** to verify that **emps_cas** was loaded into the **Casuser** caslib.
9. Run the program again. Notice that errors are produced in the log because the **emps_cas** table already exists in the CAS session. To avoid the error and replace the existing table, add the REPLACE option in the LOAD statement. Run the program again and confirm that it was successful.

```
proc casutil;
    load data=pvbase.employees outcaslib="casuser"
         casout="emps_cas" replace;
quit;
```

10. You can programmatically verify that **emps_cas** was loaded into the **Casuser** caslib by using the LIST statement with the TABLES option in PROC CASUTIL. Enter the following code and run the step.

    Note: This step will list the in-memory tables in the active caslib. If you would like to view tables in a different caslib, add the INCASLIB= option in the LIST statement.

```
proc casutil;
    list tables;
quit;
```

Table Information for Caslib CASUSER(Stacey.Syphus@sas.com)

Table Name	Number of Rows	Number of Columns	Indexed Columns	NLS encoding	Created	Last Modified	Promoted Table	Repeated Table	View	Compressed
EMPS_CAS	1048	10	0	utf-8	2020-06-23T17:51:46-04:00	2020-06-23T17:51:46-04:00	No	No	No	No

11. To produce a report with the properties of the **employee_cas** table, replace the LIST statement with the following CONTENTS statement. Run the step and view the results.

```
proc casutil;
    contents casdata="emps_cas" incaslib="casuser";
quit;
```

The CASUTIL Procedure

Table Information for Caslib CASUSER(student)

Table Name	Number of Rows	Number of Columns	Indexed Columns	NLS encoding	Created	Last Modified	Promoted Table	Repeated Table	View	Compressed
EMPS_CAS	1048	10	0	utf-8	2020-06-17T20:13:31-04:00	2020-06-17T20:13:31-04:00	No	No	No	No

Detail Information for emps_cas in Caslib CASUSER(student).

Node	Number of Blocks	Active Blocks	Rows	Data size	Variable Data size	Blocks Mapped	Memory Mapped	Blocks Unmapped	Memory Unmapped	Blocks Allocated	Memory Allocated	Index Size	Compressed Size	Compression Ratio
ALL	2	2	1048	159296	0	0	0	0	0	2	159296	0	0	0

Column Information for EMPS_CAS in Caslib CASUSER(student)

Column	Label	Type	Length	Format Name	Formatted Length	Format Width	Format Decimal
Employee_ID	Employee_ID	double	8	BEST	12	0	0
Employee_Name	Employee_Name	char	35	$	35	35	0
Country	Country	char	2	$	2	2	0

Note: SAS Viya has additional data types. The data type DOUBLE maps to a SAS numeric data type. SAS Viya also supports CHAR, which is the SAS character fixed-width data type, and the VARCHAR data type, which is a variable-length character field.

12. (Optional) You can use the CONTENTS procedure to generate a similar report of tables in the **Casuser** caslib. Highlight and submit the PROC CONTENTS step and review the log and the results.

```
proc contents data=casuser._all_ nods;
run;

proc contents data=casuser.emps_cas varnum;
run;
```

End of Demonstration

2.06 Activity (REQUIRED)

Open **startup.sas** from the **PGVY35** folder and perform the following tasks:

1. Highlight all of the code (Ctrl+A) and copy it to the clipboard (Ctrl+C).
2. Click **Options** ⇨ **Autoexec File**.
3. Paste the program code into the box (Ctrl+V). Click the **Run** and check the log to ensure that the program ran without error.
4. Click **Save**. Close the **startup.sas** program.
5. View the Libraries section of the navigation pane. Is **Casuser** listed?

Note: Autoexec.sas executes automatically every time that you sign in to SAS Studio.

50

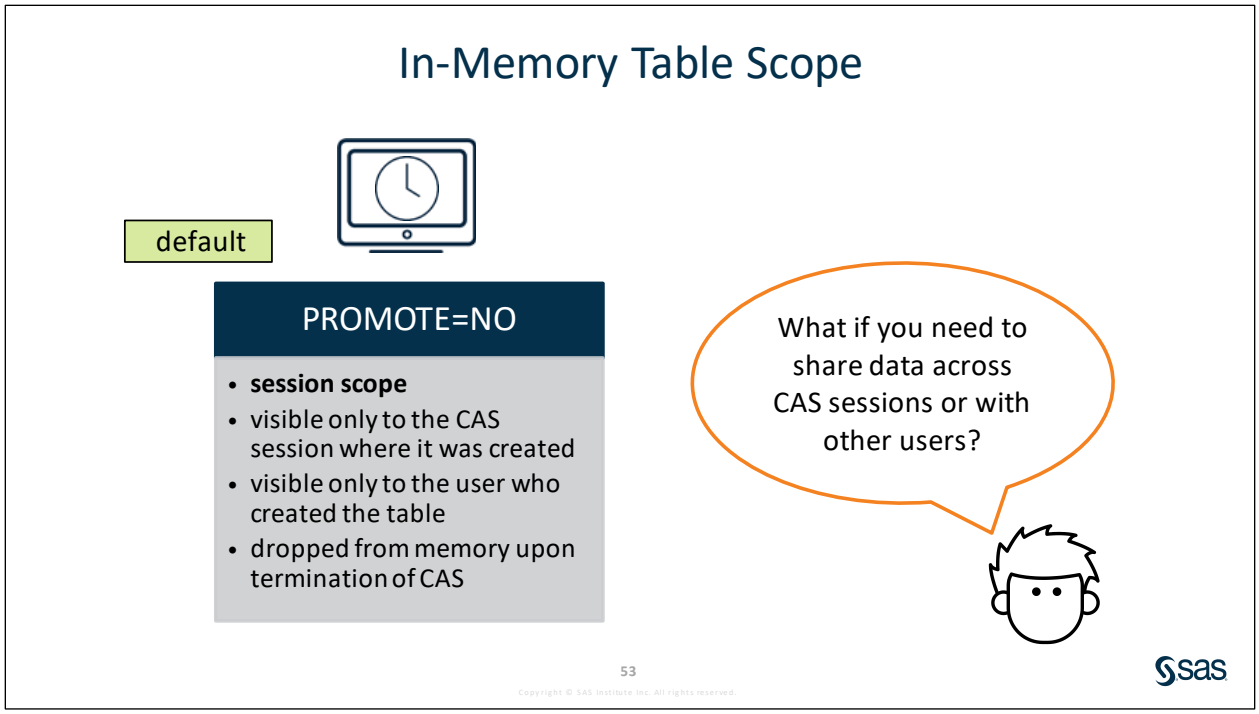
Loading Client-Side Data Source Files

Microsoft Excel or comma-separated-values (.CSV) files

```
PROC CASUTIL;
     LOAD FILE="filename.ext"
          CASOUT="target-table-name"
          <OUTCASLIB="caslib">
          <PROMOTE | REPLACE> <option(s)>;
QUIT;
```

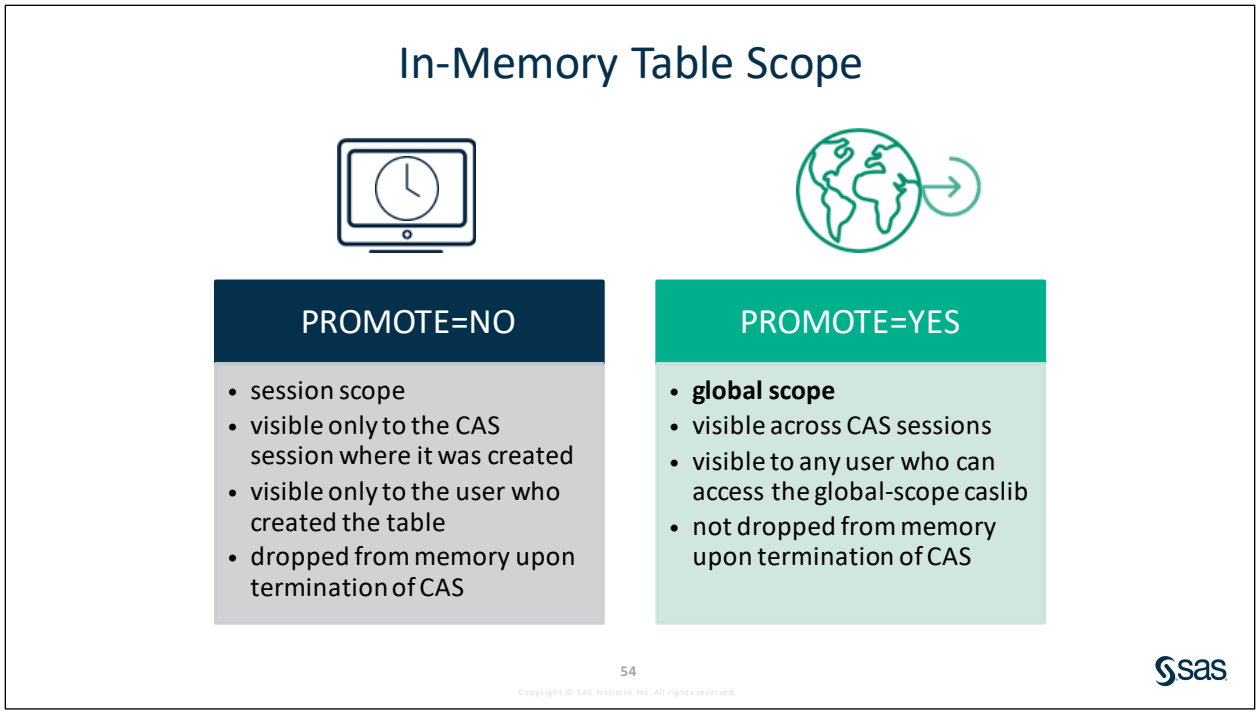
52

If you want to load a data source file that is not from SAS, such as Excel or CSV, you use the FILE= option instead of the DATA= option. Notice that the name of the file is in quotation marks, which is also true for all other option values.



By default, in-memory tables have the PROMOTE= attribute set to NO. In other words, the tables are session scope. A session-scope table is accessible only in the CAS session where it was created, and it is visible only to the user who created it. Session-scope tables are useful for ad hoc data access and analysis because they do not require access control checks or locking for concurrent access.

A session-scope in-memory table exists only for the duration of the session. When the CAS session ends, the table is dropped.



You must promote a table if you need to share data. A promoted table has global scope. You can promote a table when you load a file into memory or promote an in-memory session table.

After a session-scope table is promoted, it is visible across CAS sessions. If the global-scope table is stored in a caslib that is shared by others, anyone who has access to the global-scope caslib can access the table. There is only one copy of the table, so the administrator controls user access permissions and locking for concurrent access.

Unlike session-scope tables, global-scope tables are not dropped from memory when a CAS session ends. The table is still available to other sessions, and it will be available in the next CAS session that the user starts.

2.07 Activity

Open **pv02a07.sas** from the **activities** folder and perform the following tasks:

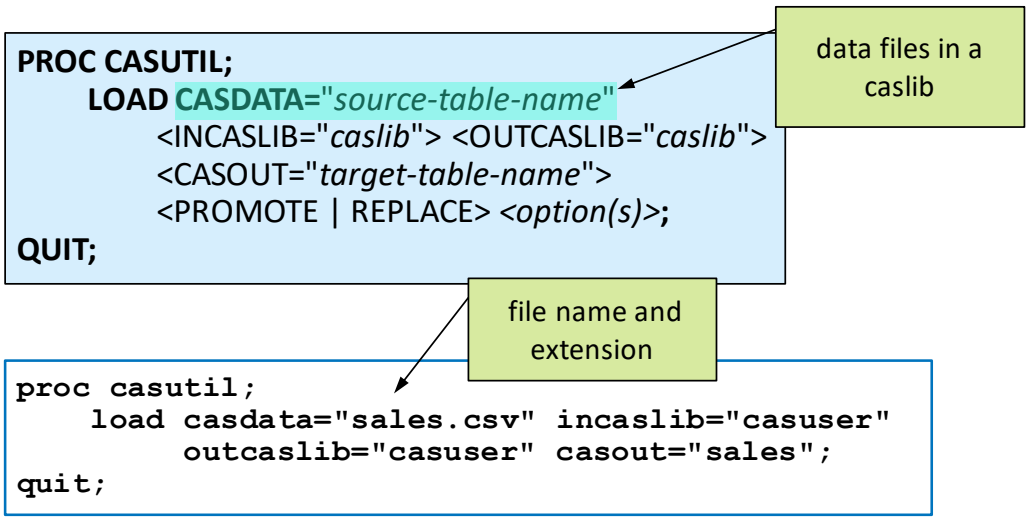
1. In PROC CASUTIL, the first LOAD statement loads **pvbase.employees** to an in-memory table named **employees**. The second LOAD statement loads the **customers.xlsx** file to an in-memory table named **customers**.
2. Add the PROMOTE option at the end of the first LOAD statement.

```
load data=pvbase.employees casout="employees" promote;
```

3. Run the program. In the Libraries section of the navigation pane, confirm that both tables are available in **Casuser**.
4. Click **Options** ⇨ **Reset SAS Session**. The SAS session restarts and the autoexec file runs. Which tables are available in **Casuser**?

55

Loading Server-Side Data Source Files

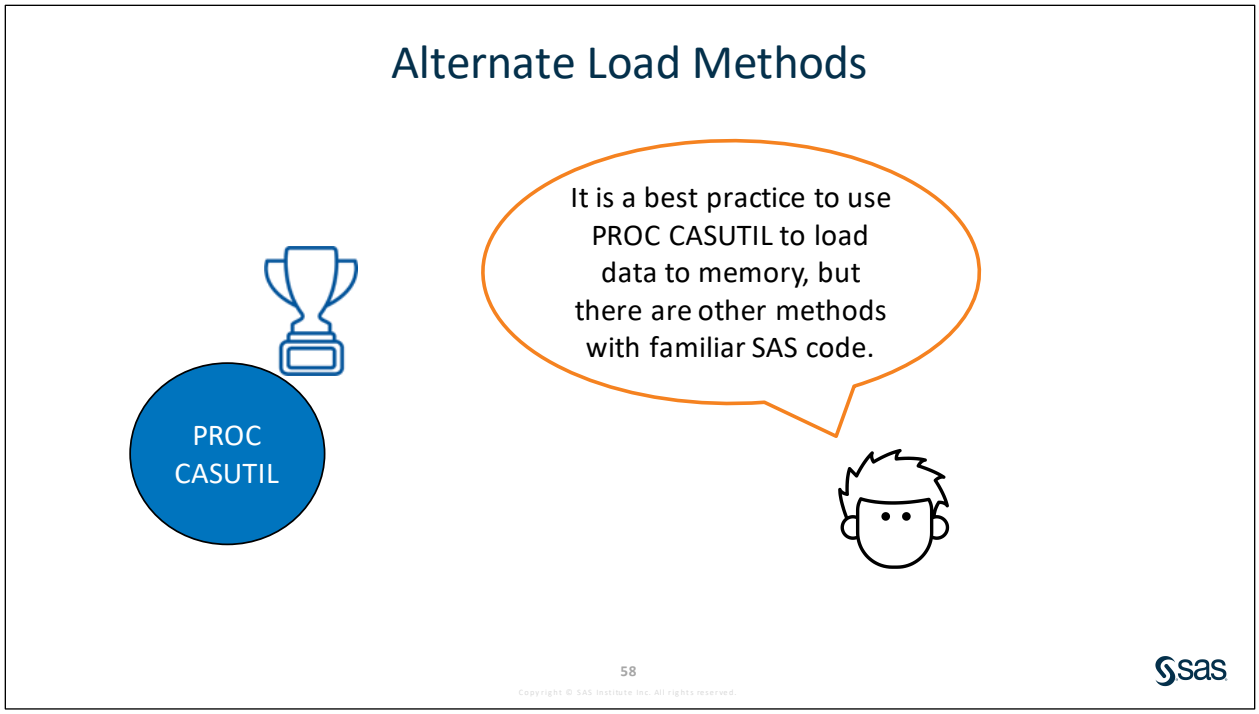


57

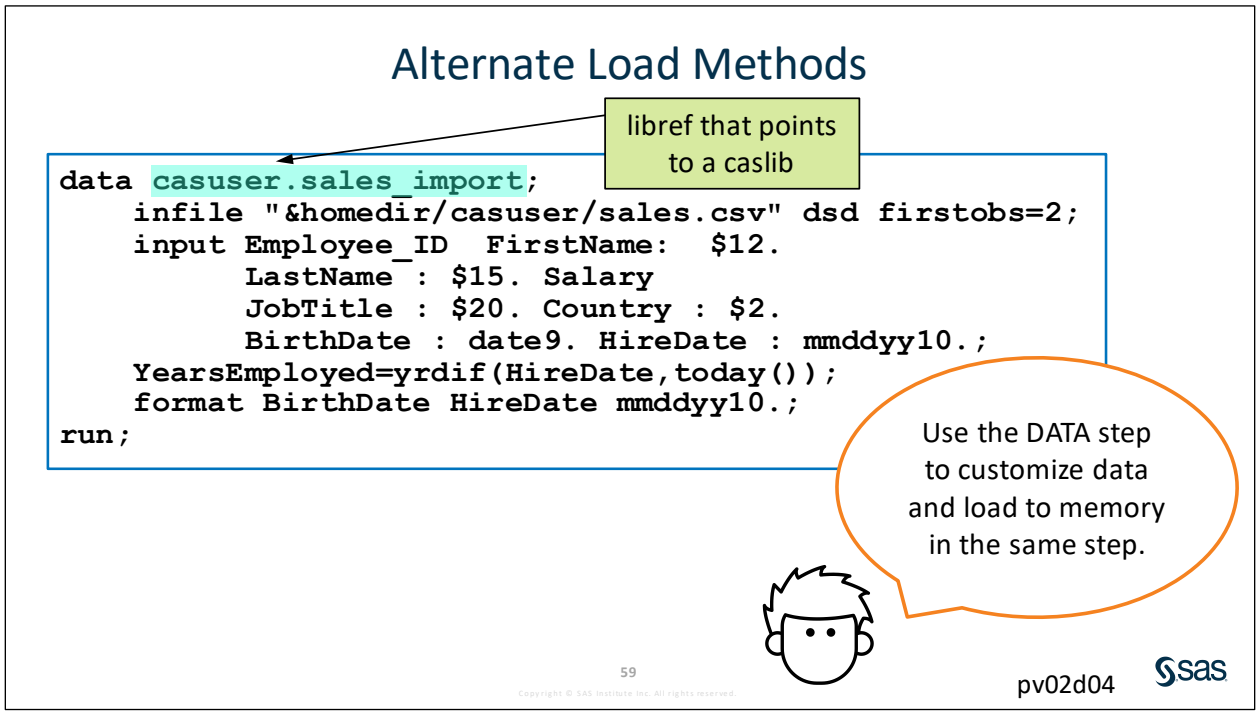
What if you need to load a server-side data source? Or, in other words, what if you need to load a data file that is in a caslib? Use PROC CASUTIL's LOAD statement with the CASDATA= option. Be sure to specify the file name, including the file extension, in quotation marks.

The CASUTIL procedure always uses the active caslib. As a best practice, you should always specify the caslib explicitly with the INCASLIB= and OUTCASLIB= options.

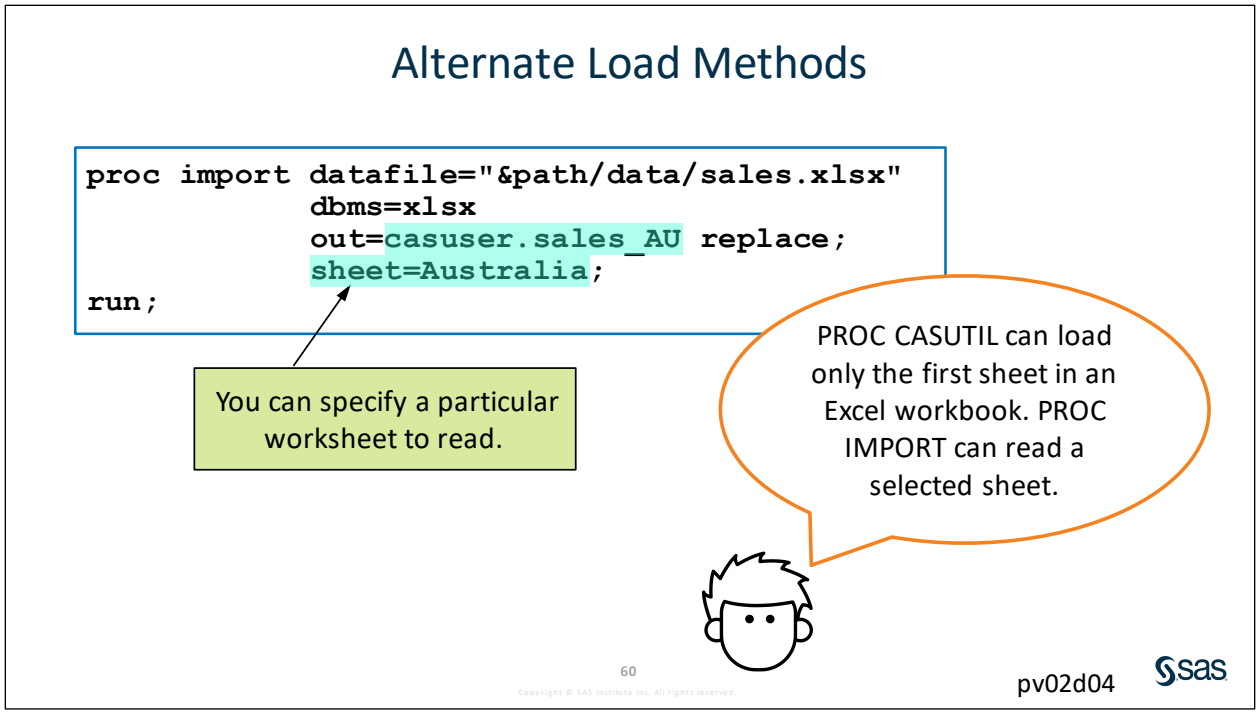
By default, the in-memory table will have the same name as the original file, but you can use the CASOUT= option to specify a different name.



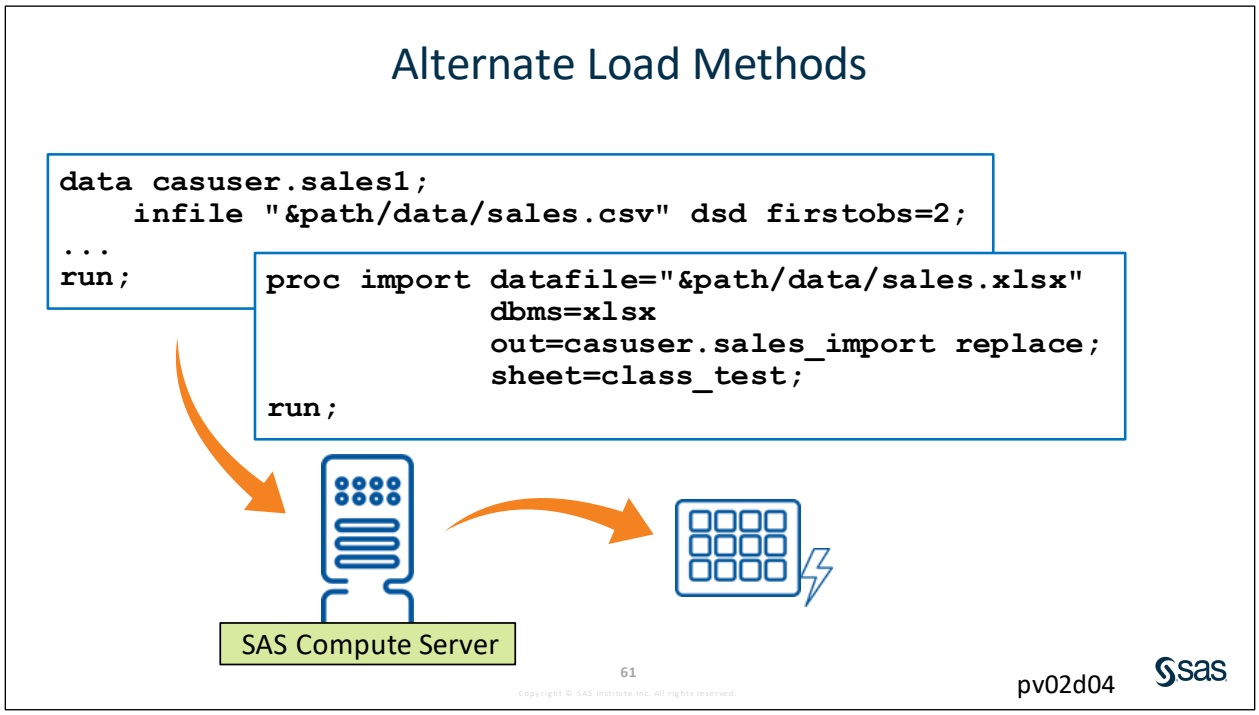
In general, PROC CASUTIL is the easiest and most efficient way to load tables into memory. However, there are other methods that use familiar syntax and provide different options compared to PROC CASUTIL.



The DATA step enables you to customize the import of a text file or manipulate an existing SAS data set. You can take advantage of all the great features that the DATA step provides and then load the result in memory, all in the same step. Simply direct your output table to a caslib in the DATA statement.



PROC CASUTIL loads only the first worksheet within an Excel workbook. You can also use a PROC IMPORT step to read an Excel workbook, and you can use the SHEET= option to read a specific worksheet. If you specify a caslib library reference in the OUT= option, the data will be loaded into CAS.



In both of these examples, the code is processed on the Compute Server, but the output data is loaded to an in-memory table.

Practice

Required setup: Everyone needs to do this once.

All the remaining demos and practices rely on having your SAS library and CAS session already established. Activity 2.06 provided the instructions to set up an ***autoexec file*** in SAS Studio to accomplish this automatically. If you did not complete that activity, here are the instructions required to do it now:

1. Open the **startup.sas** program in the **PGVY35** folder. Highlight all the code and copy it (Ctrl+C) to the clipboard.
2. In SAS Studio, click **Options** ⇨ **Autoexec File**.
3. Paste (Ctrl+V) the program code into the box.
4. Click **Run** and check the log to ensure that the program ran without errors.
5. Click **Save**, and the window closes. Your **Autoexec.sas** program executes automatically every time that you sign in to SAS Studio.
6. Close the **startup.sas** program.

Level 1

1. Loading a Client-Side SAS Data Set and Excel File into CAS Using a Snippet

a. Open the **Load data to caslib** snippet from the **SAS Viya Cloud Analytic Services** snippet category.

b. Modify the first PROC CASUTIL step in the snippet to load the server-side **products.xlsx** file into the **Casuser** caslib.

1) Highlight *pathToClientFile* in the FILE= option. In the Explorer section, navigate to **server** ⇨ **Home** ⇨ **casuser**. Right-click **products.xlsx** and select **Insert as path**.
2) Change *mycaslib* to **casuser** in the OUTCASLIB= option.
3) Change *tableNameForLoadedFile* to **products** in the CASOUT= option.
4) Add the PROMOTE option so that the in-memory table will have global scope.
5) Change **run** to **quit**.

c. Modify the second PROC CASUTIL step in the snippet to load the **pvbase.qtr_sales** SAS data set into the **Casuser** caslib.

1) Change *library.tablename* to **pvbase.qtr_sales** in the DATA=option.
2) Change *mycaslib* to **casuser** in the OUTCASLIB= option.
3) Change *targetTableName* to the **qsales** in the CASOUT= option.
4) Change **run** to **quit**.

d. Highlight and delete the last comment block and PROC CASUTIL step. Submit the modified program.

Note: The path might be different in your environment.

```
proc casutil;
    load file="/home/student/casuser/products.xlsx"
         outcaslib="casuser" casout="products" promote;
quit;

proc casutil;
    load data=pvbase.qtr_sales outcaslib="casuser"
         casout="qsales";
quit;
```

e. Add a PROC CASUTIL step and the LIST TABLES statement to confirm that the tables were successfully loaded into the **Casuser** caslib. Answer the following questions:

1) How many rows are in the **products** table?
2) Is the **qsales** table session scope or global scope?

Level 2

2. Loading and Promoting a Server-Side File into CAS

a. Open the **Load data to caslib** snippet from the **SAS Viya Cloud Analytic Services** snippet category. Highlight and delete the first two PROC CASUTIL steps and comment blocks.

b. Modify the last CASUTIL step to load **sales.sas7bdat** from the **Casuser** caslib. Name the in-memory table **sales** and promote it to global scope.

c. Add a CONTENTS statement to view the table and column attributes. Run the program. Notice that **Birth Date** and **Hire Date** have different date formats applied.

d. The ALTERTABLE statement in PROC CASUTIL can be used in PROC CASUTIL to change formats and labels in in-memory tables. Add a second PROC CASUTIL step. Include the ALTERTABLE statement to apply the DATE7. format to the **Hire Date** column. Add a CONTENTS statement in the second PROC CASUTIL step.

```
altertable casdata="table"
           columns={{name="<colname>" format="format."}};
```

e. Submit the step and view the results. Confirm that the DATE format is applied to the **Hire Date** column. Is the **sales** table still promoted after being altered?

Challenge

3. Loading a CSV File That Contains No Header Information into CAS

This step loads the **sales.csv** file into CAS. The code uses the IMPORTOPTIONS= option in the LOAD statement to specify the file type that you are loading (**filetype="csv"**). The encoding value for the source file is specified in the ENCODING= suboption.

a. Open **pv02p03.sas** from the **practices** folder. Highlight and submit the first PROC CASUTIL step and confirm that the table was stored in the **Casuser** caslib. Notice that column names are **Var1**-**Var8** and that, although the last two columns are date values, the column type is VARCHAR.

b. Modify the second PROC CASUTIL step to specify the names of the variables using the VARS= suboption in the IMPORTOPTIONS= option. The values that are needed for the first column have been specified. Modify the rest to match the following attributes to name the columns, labels, and data types:

Column Name Name=" "	Label Label=" "	Data Type Type=" "
name="Id"	label="Identification Number"	type="varchar"
First	First Name	VARCHAR
Last	Last Name	VARCHAR
Salary	Annual Salary	DOUBLE
JobTitle	Employee Job Title	VARCHAR
Country	Employee Country	VARCHAR
DOB	Birth Date	Do not use the type=" "
HireDate	Date of Hire	Do not use the type=" "

c. Submit the PROC CASUTIL step. Note that the type of **DOB** and **HireDate** is VARCHAR.

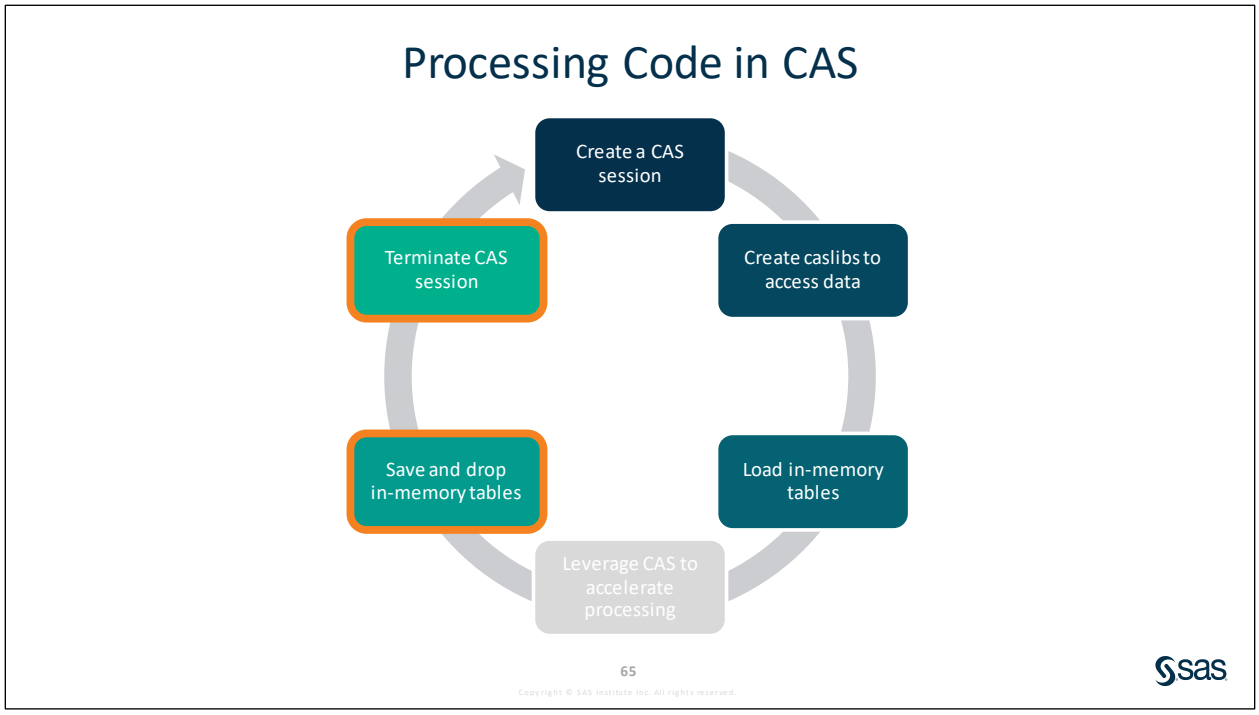
d. The DATA step can process the in-memory data to modify the column type for **DOB** and **HireDate**. You can also perform other data manipulation tasks in-memory. Modify the DATA step code to do the following:

1) Use the DATE9. informat in the INPUT function to create a new column, **Birth_Date**, that converts **DOB** to a SAS date value.
2) Use the MMDDYY10. informat in the INPUT function to create a new column, **Hire_Date**, that converts **HireDate** to a SAS date value.
3) Modify the FORMAT statement to format **Birth_Date** with the DATE9. format and **Hire_Date** with the MMDDYY10. format.
4) Add a DROP statement to drop **DOB** and **HireDate**.
5) Highlight and submit the OPTIONS statement through the last PROC CASUTIL step and verify the descriptor portion of the new in-memory table, **casuser.salesimport_final**.

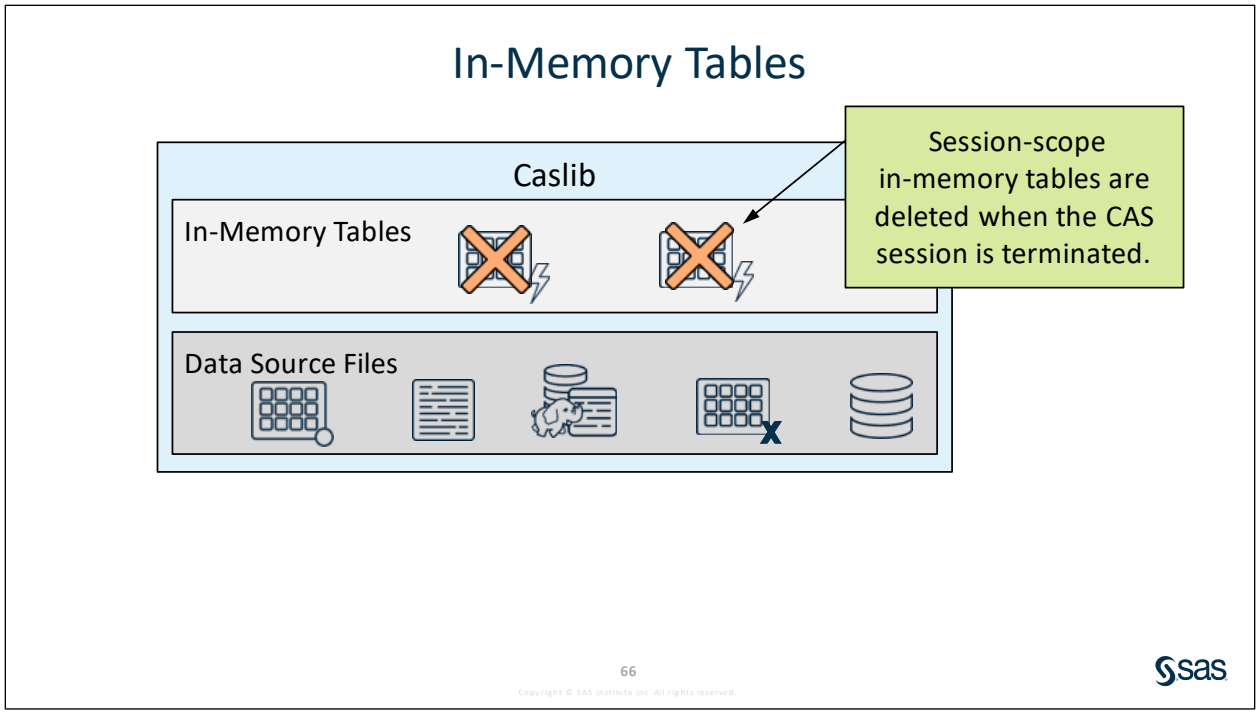
   What is the data type for **Birth_Date** and **Hire_Date**?

End of Practices

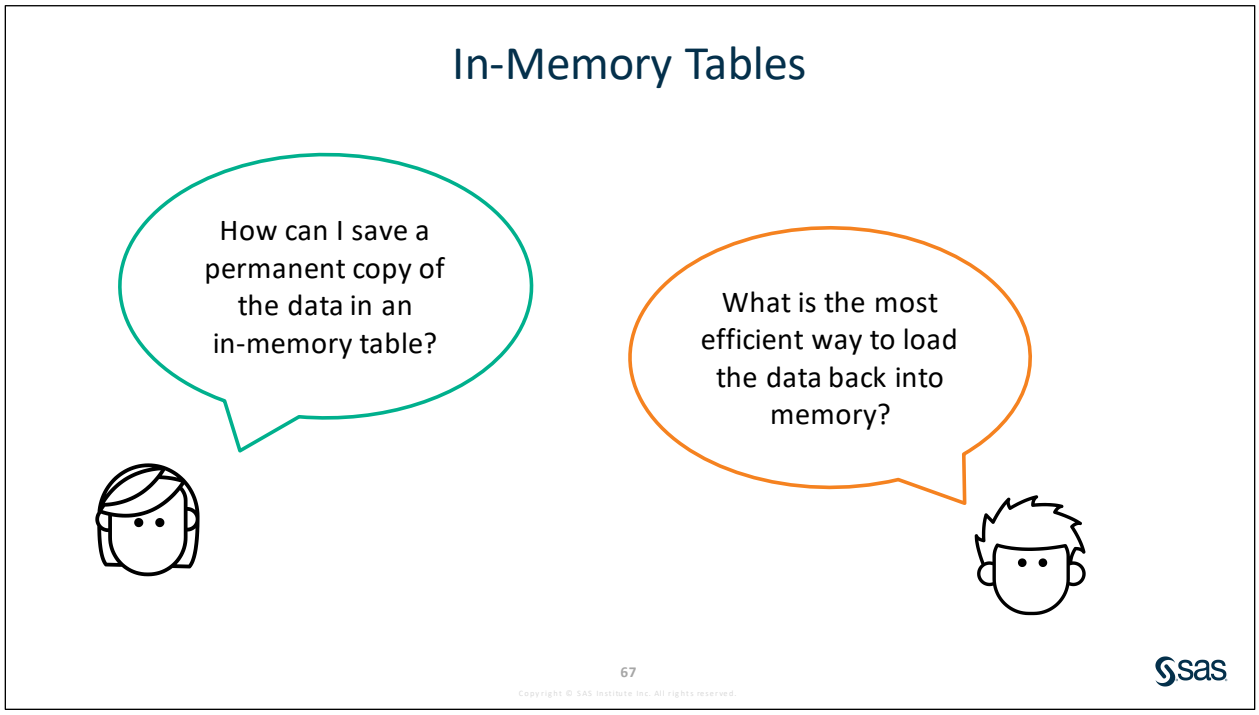
2.3 Saving and Dropping In-Memory Tables



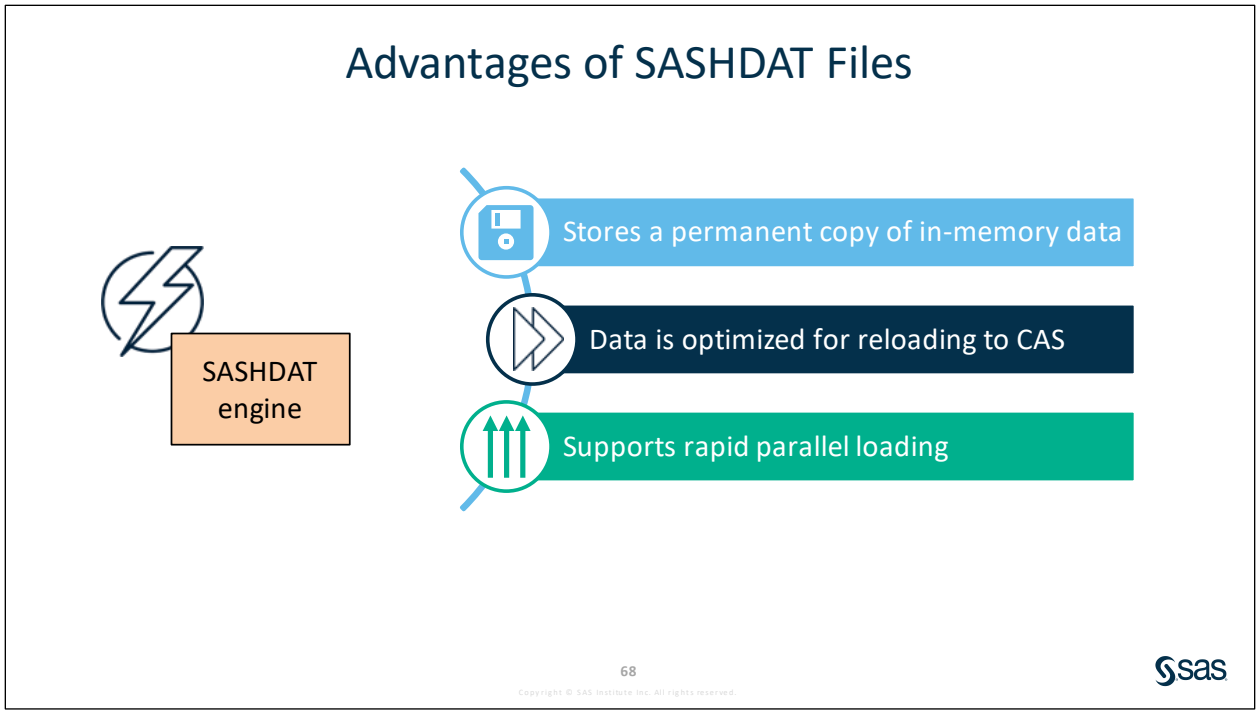
When you are done processing your in-memory tables, it is a good practice to tidy up a bit. This includes possibly saving your tables and then dropping them from memory. And the final step in the cycle would be to terminate the CAS session.



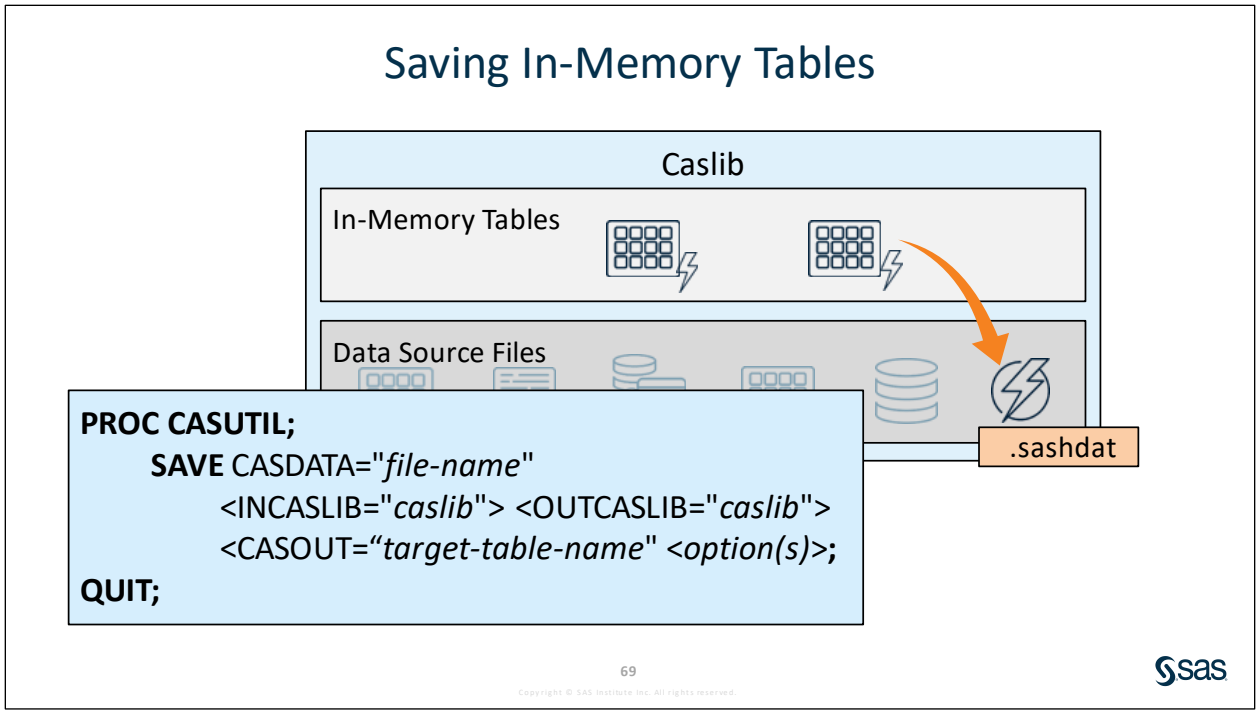
When the CAS session ends, tables that have not been promoted, or session-scope tables, are automatically deleted.



But what if I have modified that in-memory table in some way, and I would like to save a permanent copy? And is there a way to optimize the storage of that saved copy so that it can rapidly be loaded later into memory?



The SASHDAT engine enables you to store data in a format that is optimized for distributing data in CAS to the available worker nodes. It supports parallel loading, meaning that data can be loaded simultaneously to the multiple workers, which can significantly reduce load time.



If you want to remove the table from memory but save the data, you can save it as a SASHDAT file in the caslib's data source files. A SASHDAT file is a permanent copy of an in-memory table that can be quickly loaded into memory in the future, without reloading the original data from the client.

The SAVE statement can also be used to create other file types. Use the EXPORTOPTIONS= option to specify the file type:

EXPORTOPTIONS={FILETYPE="AUTO" | "BASESAS" | "DTA" | "EXCEL" | "JMP" | "ORC" | "SPSS" | "XLS", *file-type-specific-parameters*}

Saving a SASHDAT File in a Caslib

Scenario

Save a SASHDAT file in a caslib for future use.

Program

- **pv02d05**

Syntax

```
PROC CASUTIL;
     SAVE CASDATA="file-name" <INCASLIB="caslib"><CASOUT="target-table-name"
          <OUTCASLIB="caslib"> <option(s)>;
QUIT;
```

Notes

- The CASDATA= option specifies the name of the in-memory table to save. This is required.
- The CASOUT= option specifies an alternate name for the file. By default, a **.sashdat** suffix is used when you save to a path-based caslib. If you want to save a CSV file, use the .csv suffix.
- The OUTCASLIB= option specifies where the SASHDAT file will be created. If the option is not specified, the table is loaded into the active caslib.

Demo

Open **pv02d05.sas** from the **demos** folder and perform the following tasks:

Note: If you have not set up the autoexec file in SAS Studio, open and submit **startup.sas** first.

1. The CASUTIL step loads the **pvbase.employees** SAS data set to an in-memory table named **employees**. If the table is already in-memory, it will be replaced. Run the program and confirm in the log that the table was successfully added to the **Casuser** caslib.
2. Return to the Code tab and place the cursor at the end of the program.
3. In the Snippets section of the navigation pane, expand **SAS Viya Cloud Analytic Services**, and then right-click **Save table to caslib** and select **Insert**. Modify the code as follows:
   a. Change "*sourceTableName*" to **"employees"** in the CASDATA= option.
   b. Change "*sourceCaslib*" to **"casuser"** in the INCASLIB= option.
   c. Change "*targetCaslib*" to **"casuser"** in the OUTCASLIB= option.
   d. Change "*file-name*" to **"emps_hd"** in the CASOUT=option.

      Note: If you do not provide a file extension, a .sashdat file will be created. You can provide other file extensions such as .xlsx, .csv, or .sas7bdat.

   e. Add a LIST statement to view the data files in the **Casuser** caslib.

```
proc casutil;
    save casdata="employees" incaslib="casuser"
         outcaslib="casuser" casout="emps_hd";
    list files;
quit;
```

4. Submit the modified code. Examine the log to confirm that the **emps_hd.sashdat** file was saved to the **Casuser** caslib.

CAS File Information

Name	Permission	Owner	Group	Encryption Method	File Size	Last Modified (UTC)
sales.csv	-rwxr-xr-x	Stacey.Syphus@sas.com	v4e_users		10.3KB	23JUN2020:20:10:57
orders_hd.sashdat	-rwxr-xr-x	Stacey.Syphus@sas.com	v4e_users	NONE	1.6GB	23JUN2020:20:11:06
sales.sas7bdat	-rwxr-xr-x	Stacey.Syphus@sas.com	v4e_users		72.0KB	23JUN2020:20:11:07
products.xlsx	-rwxr-xr-x	Stacey.Syphus@sas.com	v4e_users		219.8KB	23JUN2020:20:11:07
customer_report.xlsx	-rw-r--r--	Stacey.Syphus@sas.com	v4e_users		24.8KB	23JUN2020:20:24:56
emps_hd.sashdat	-rwxr-xr-x	Stacey.Syphus@sas.com	v4e_users	NONE	163.8KB	23JUN2020:22:14:14

End of Demonstration

2.08 Activity

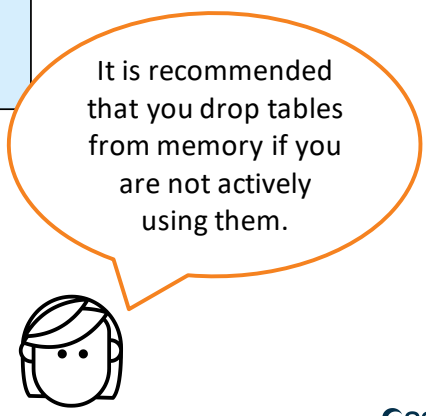
Open **pv02a08.sas** from the activities folder and perform the following tasks:

1. The first PROC CASUTIL step loads the Base SAS table **pvbase.orders** to an in-memory table named **orders_loadbase**. The table includes 24 columns and 951,669 rows.
2. The second PROC CASUTIL step loads the same data stored in a SASHDAT file to an in-memory table named **orders_loadhdat**.
3. Run the program and view the log. Examine the real time and CPU time for both load methods. Which PROC CASUTIL step loaded the **customers** data into memory the fastest?

71

Dropping In-Memory Tables

```
PROC CASUTIL;
      DROPTABLE CASDATA="table-name"
                  <INCASLIB="caslib"> <QUIET>;
QUIT;
```



73

When you are done working with in-memory tables, it is a good idea to clean up afterward. This can be done by using the DROPTABLE statement to remove tables from memory.

Dropping In-Memory Tables

```
PROC CASUTIL;
      DROPTABLE CASDATA="table-name"
                  <INCASLIB="caslib"> <QUIET>;
QUIT;
```

```
proc casutil;
    droptable casdata="employees"
              incaslib="casuser" quiet;
quit;
```

The QUIET option suppresses error messages if the specified table is not found.

74

The QUIET option is helpful to add because it suppresses error messages if the table is not found in-memory.

2.09 Activity

1. In the Snippets section of the navigation pane, open the **Delete Table or File from caslib** snippet.
2. In the Libraries section, expand **CASUSER** to view the tables currently loaded in-memory.
3. In the second PROC CASUTIL step, change *tableName* to the first table listed in **Casuser**, and change *sourceCaslib* to **Casuser**.
4. Run the step and verify that the in-memory table is no longer in **Casuser**.
5. Open and run the **Terminate CAS Session** snippet. Is **Casuser** accessible in the Libraries section?

75

Practice

Note: If you have not set up the autoexec file in SAS Studio, open and submit **startup.sas** first.

Level 1

4. Saving a Copy of the In-Memory Table as a SASHDAT File

a. Open **pv02p04.sas** from the **practices** folder. The PROC CASUTIL step deletes the in-memory sales table if it exists, and then loads the server-side Base SAS table **sales.sas7bdat** from the **Casuser** caslib. Run the step.

b. Place the cursor at the end of the program. Insert the **Save Table to caslib** snippet. Modify the SAVE statement to create a copy of the in-memory **sales** table as **sales_hd.sashdat**. Save the file in the **Casuser** caslib.

c. Add a DROPTABLE statement to remove the **sales** table from memory after the SASHDAT file has been created.

d. Add LIST statements to view the tables and files included in **Casuser**.

e. Run the program and view the **CAS File Information** report. Which file size is smaller, **sales.sas7bdat** or **sales_hd.sashdat**?

End of Practices

2.4 Solutions

Solutions to Practices

1. Loading a Client-Side SAS Data Set and Excel File into CAS Using a Snippet

```
 /* Parts a.-b. */
proc casutil;
    load file="/home/student/casuser/products.xlsx"
         outcaslib="casuser" casout="products" promote;
quit;

 /* Parts c.-d. */
proc casutil;
    load data=pvbase.qtr_sales outcaslib="casuser"
         casout="qsales";
quit;

 /* Part e. */
proc casutil;
    list tables;
quit;
```

How many rows are in the **products** table? **5504**

Is the **qsales** table session scope or global scope? **Session scope (PROMOTED=NO)**

2. Loading and Promoting a Server-Side File into CAS

```
 /* Parts a.-c. */
proc casutil;
    load casdata="sales.sas7bdat" incaslib="casuser"
         outcaslib="casuser" casout="sales" promote;
    contents casdata="sales";
quit;

 /* Parts d.-e. */
proc casutil;
    altertable casdata="sales"
               columns={{name="Hire Date" format="date7."}};
    contents casdata="sales";
quit;
```

Is the **sales** table still promoted after being altered? **PROMOTE is YES, so the table is still global scope.**

3. Loading a CSV File That Contains No Header Information into CAS

```
 /* Part a. */
proc casutil;
    load casdata="sales.csv" casout="salesimport"
         importoptions=(filetype="csv" encoding="latin1"
                        getnames="false") replace;
    contents casdata="salesimport";
quit;

 /* Parts b.-c. . */
proc casutil;
    load casdata="sales.csv" casout="salesimport"
         importoptions=(filetype="csv" encoding="latin1"
                         getnames="false"
         vars=((name="ID", label="Identification Number",
                type="varchar"),
            (name="First", label="First Name", type="varchar"),
            (name="Last", label="Last Name", type="varchar"),
            (name="Salary", label="Annual Salary",type="double"),
    (name="JobTitle",label="Employee Job Title", type="varchar"),
    (name="Country",label="Employee Country",type="varchar"),
            (name="DOB", label="Birth Date"),
            (name="HireDate",label="Date of Hire")
           )
           ) replace;
    contents casdata="salesimport";
quit;

 /* Part d. */
options msglevel=i;
data casuser.salesimport_final(promote=yes);
    set casuser.salesimport;
    Birth_Date=input(DOB,date9.);
    Hire_Date=input(HireDate,mmddyy10.);
    format Birth_Date date9. Hire_Date mmddyy10.;
    drop DOB HireDate;
    label Birth_Date="Birth Date" Hire_Date="Date of Hire";
run;

options msglevel=n;
proc casutil;
    contents casdata="salesimport_final";
quit;
```

What is the data type for **Birth_Date** and **Hire_Date**? **Double (numeric)**

4. Saving a Copy of the In-Memory Table as a SASHDAT file

```
 /* Part a. */
proc casutil;
    droptable casdata="sales" quiet;
    load casdata="sales.sas7bdat" incaslib="casuser"
         outcaslib="casuser" casout="sales";
quit;

 /* Parts b.-d. */
proc casutil;
    save casdata="sales" incaslib="casuser" outcaslib="casuser"
         casout="sales_hd";
    droptable casdata="sales" quiet;
    list tables;
    list files;
quit;
```

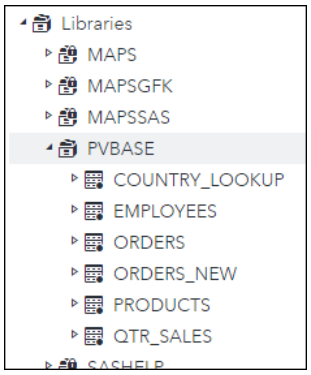
Which file size is smaller, **sales.sas7bdat** or **sales_hd.sashdat**? **Sales_hd.sashdat is smaller.**

End of Solutions

Solutions to Activities and Questions

2.01 Activity – Correct Answer

Why are the Microsoft Excel and text files from the data folder not visible in the SAS library?



The **pvbase** library uses the Base engine, so the Base SAS tables are the only files recognized and visible in the SAS library.

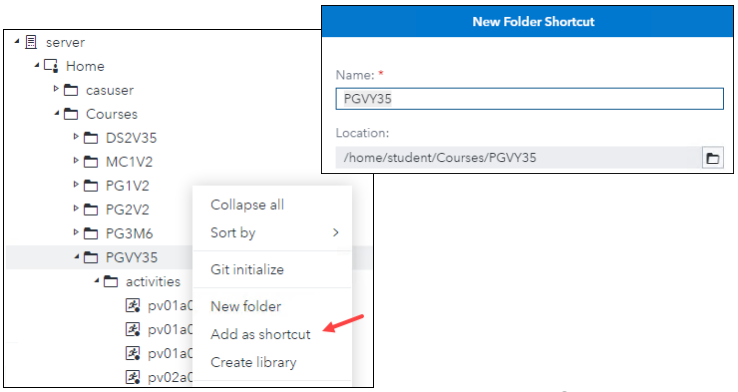
7

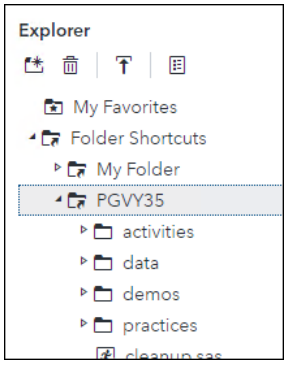
sas

2.02 Activity – Correct Answer

creating the **PGVY35** folder shortcut

accessing the **PGVY35** folder shortcut





9

sas

continued...

2.03 Activity – Correct Answer

2. Which caslib is analyzed in the report? How many data source files are in the caslib?

Casuser is analyzed because it is the active caslib.

Caslib Information	
Library	CASUSER(student)
Source Type	PATH
Description	Personal File System Caslib
Path	/opt/sas/viya/config/data/cas/default/casuserlibraries/student/
Session local	No
Active	Yes
Personal	Yes
Hidden	No
Transient	Yes

There are four data source files in **Casuser**.

CAS File Information

Name	Permission	Owner	Group	Encryption Method	File Size	Last Modified (UTC)
orders_hd.sashdat	-rwxr-xr-x	cas	sas	NONE	1.6GB	17JUN2020:18:00:00
products.xlsx	-rwxr-xr-x	cas	sas		560.8KB	17JUN2020:18:00:07
sales.csv	-rwxr-xr-x	cas	sas		10.3KB	17JUN2020:18:00:07
sales.sas7bdat	-rwxr-xr-x	cas	sas		72.0KB	17JUN2020:18:00:07

28

2.03 Activity – Correct Answer

3. Change the FILES options to TABLES. Run the program and view the log. How many tables are in the caslib?
There are no in-memory tables currently in Casuser.

```
NOTE: No tables are available in caslib CASUSER(student)
of Cloud Analytic Services.
```

29

2.04 Activity – Correct Answer

Is the Personal attribute Yes or No? **No**

```
caslib pvcas path="/home/student/Courses/PGVY35/data"
              libref=pvcas;

proc casutil;
    list files;
quit;

caslib pvcas clear;
```

Caslib Information	
Library	PVCAS
Source Type	PATH
Path	/home/student/Courses/PGVY35/data/
Session local	Yes
Active	Yes
Personal	No
Hidden	No
Transient	No

All caslibs other than **Casuser** are Personal=No.

37

continued...

2.05 Activity – Correct Answer

```
libname pvbase "/home/student/Courses/PGVY35/data";
```

Directory	
Libref	PVBASE
Engine	V9
Physical Name	/home/student/Courses/PGVY35/data

reads .sas7bdat files with the default Base (V9) engine and processes tables on the Compute Server

#	Name	Member Type	File Size	Last Modified
1	ALLCOSTS	DATA	704KB	05/08/2020 14:09:45
2	CUSTOMERS	DATA	2GB	04/28/2020 17:24:15
3	EMPLOYEES	DATA	320KB	04/28/2020 17:20:34
4	PRODUCTS	DATA	5MB	04/28/2020 17:20:57

39

2.05 Activity – Correct Answer

```
caslib pvcas path="/home/student/Courses/PGVY35/data"
           libref=pvcas;
```

provides access to all data source files and processes in-memory tables in CAS using the CAS engine

Directory	
Libref	PVCAS
Engine	CAS
...	
Session Name	MYSESSION
Server Host	server.demo.sas.com
Server Port	5570
Caslib	PVCAS

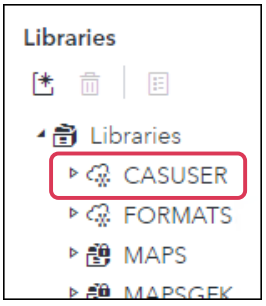
No tables are loaded into memory yet.

```
88 proc contents data=pvcas._all_ nods;
89 run;
NOTE: No tables are available in caslib
PVCAS of Cloud Analytic Services.
WARNING: No matching members in directory.
```

40

2.06 Activity – Correct Answer

View the Libraries section of the navigation pane. Is **Casuser** listed? **Yes**



This autoexec program
- creates the macro variable **Path** that stores the full path of the **PGVY35** folder
- assigns the **pvbase** SAS library to access SAS data sets in the **data** folder
- starts a CAS session named **Mysession**
- assigns **Casuser** as the active caslib
- assigns librefs for all existing caslibs, including **Casuser**.

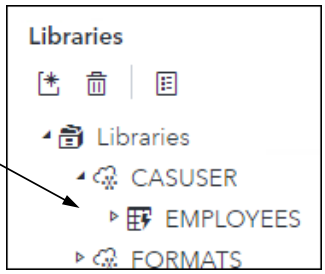
Note: All subsequent demos, activities, and practices require this saved autoexec file.

51

2.07 Activity – Correct Answer

Which tables are available in **Casuser**?

Employees was promoted to a global-scope table, so it is automatically available in each new CAS session.



Customers was a local-scope table, so it is not available after the CAS session ends.

56

2.08 Activity – Correct Answer

Which PROC CASUTIL step loaded the **customers** data into memory the fastest?

```
NOTE: PROCEDURE CASUTIL used (Total process time):
real time 7.62 seconds
cpu time 3.79 seconds
```

```
proc casutil;
    load data=pvbase.orders
         casout="orders_loadbase" replace;
quit;
```

```
NOTE: The Cloud Analytic Services server
processed the request in 0.005901 seconds.
```

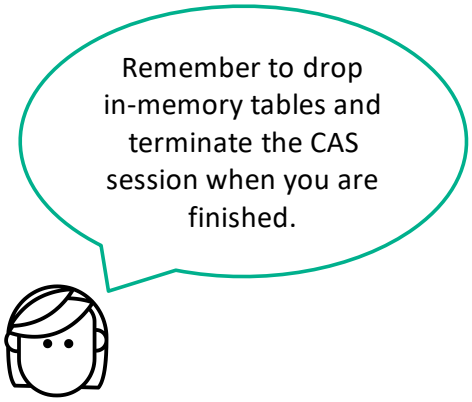
```
proc casutil;
    load casdata="orders_hd.sashdat"
         casout="customers_hd" replace;
quit;
```

72

2.09 Activity – Correct Answer

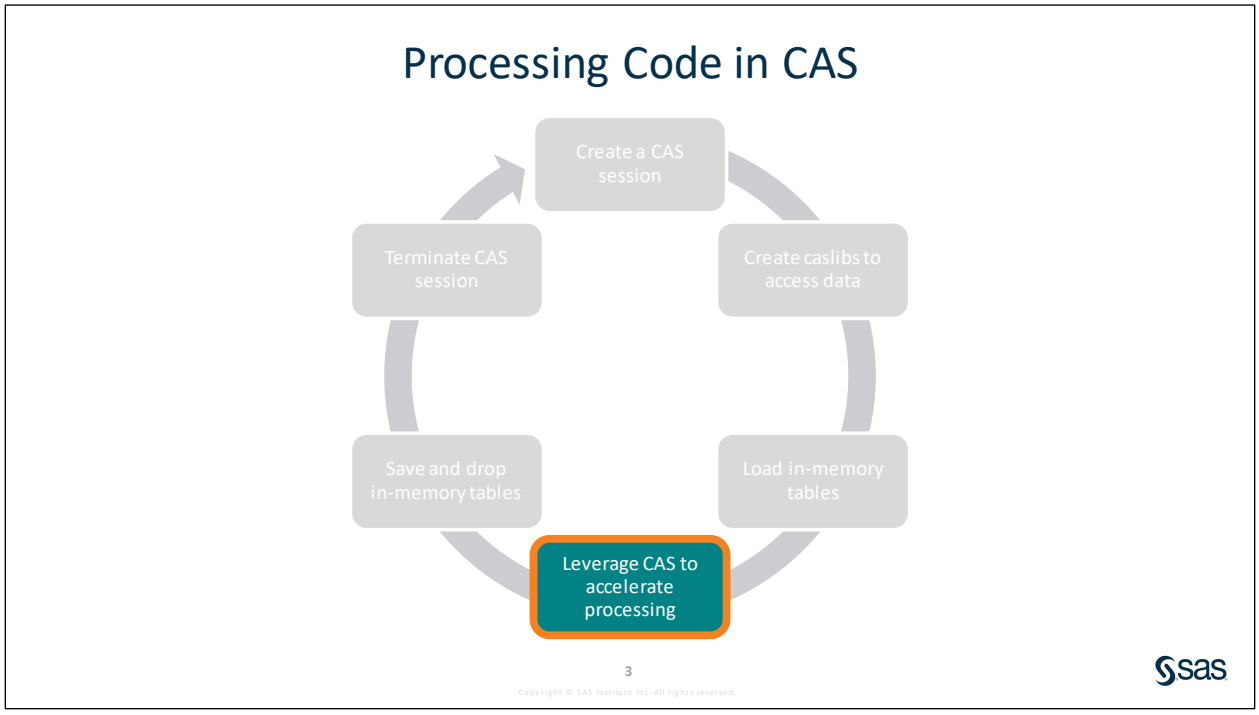
Open and run the **Terminate CAS Session** snippet. Is **Casuser** accessible in the Libraries section? **No**

Libraries defined on the Compute Server are still available.

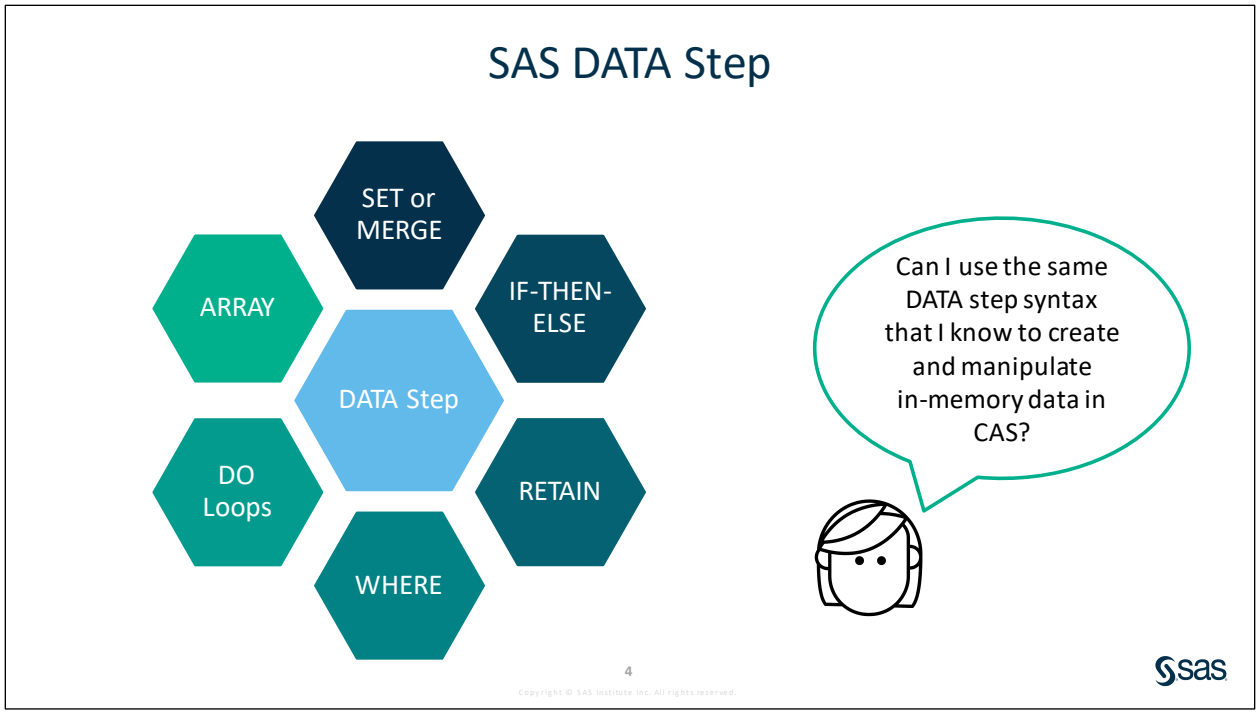


Lesson 3 Modifying Base SAS® Programs to Run in SAS Cloud Analytic Services (CAS)

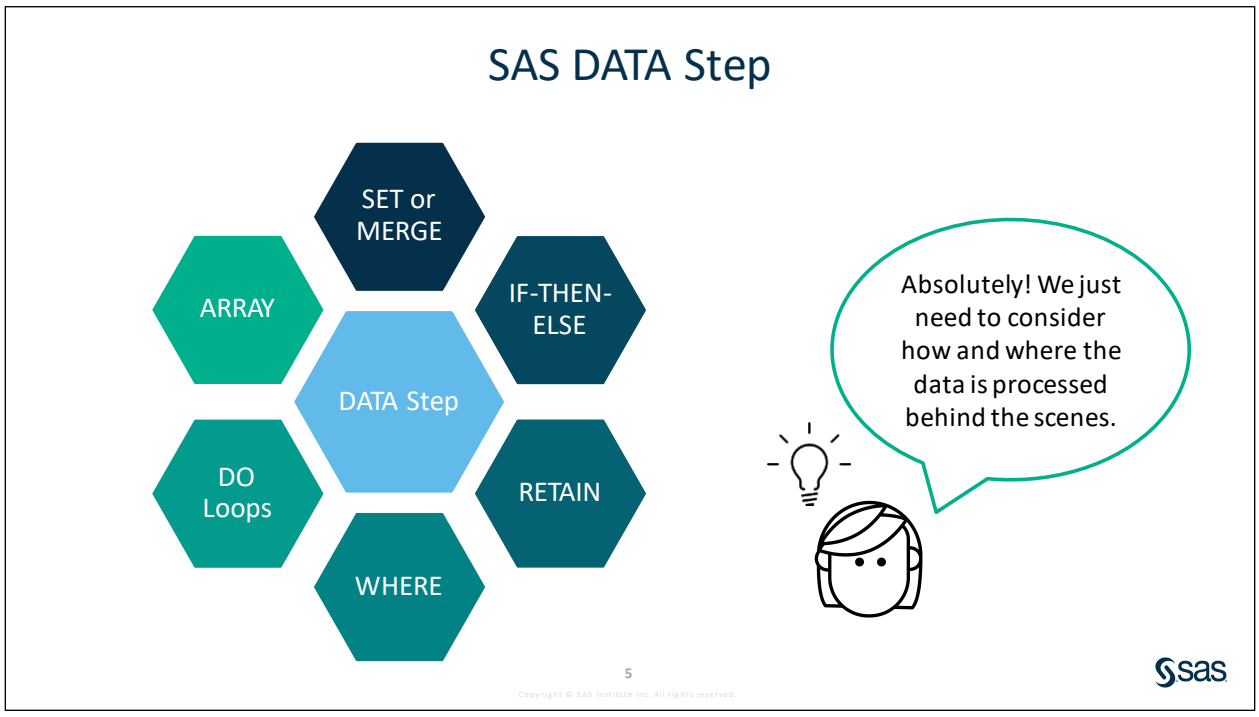
3.1 Modifying DATA Step Code for CAS



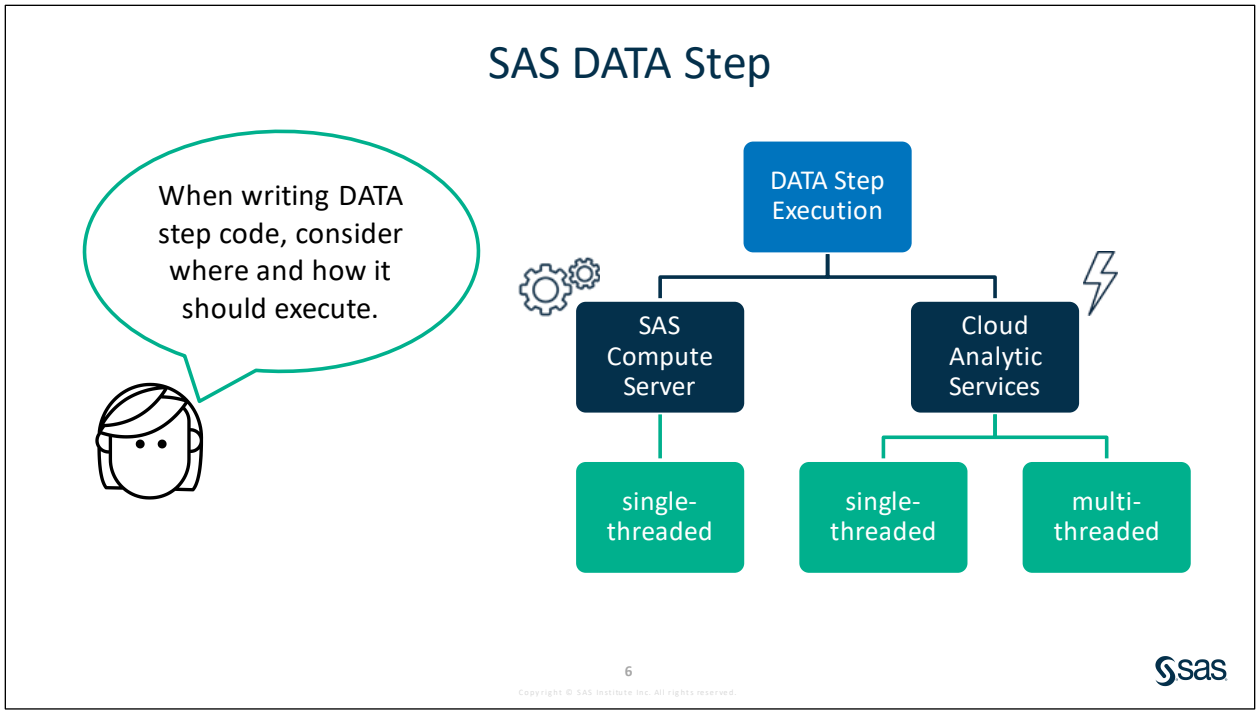
Now that we have created a CAS session and loaded data into memory, we move our attention to the rest of our SAS code. What changes, if any, are required to ensure that code runs in CAS on in-memory tables?



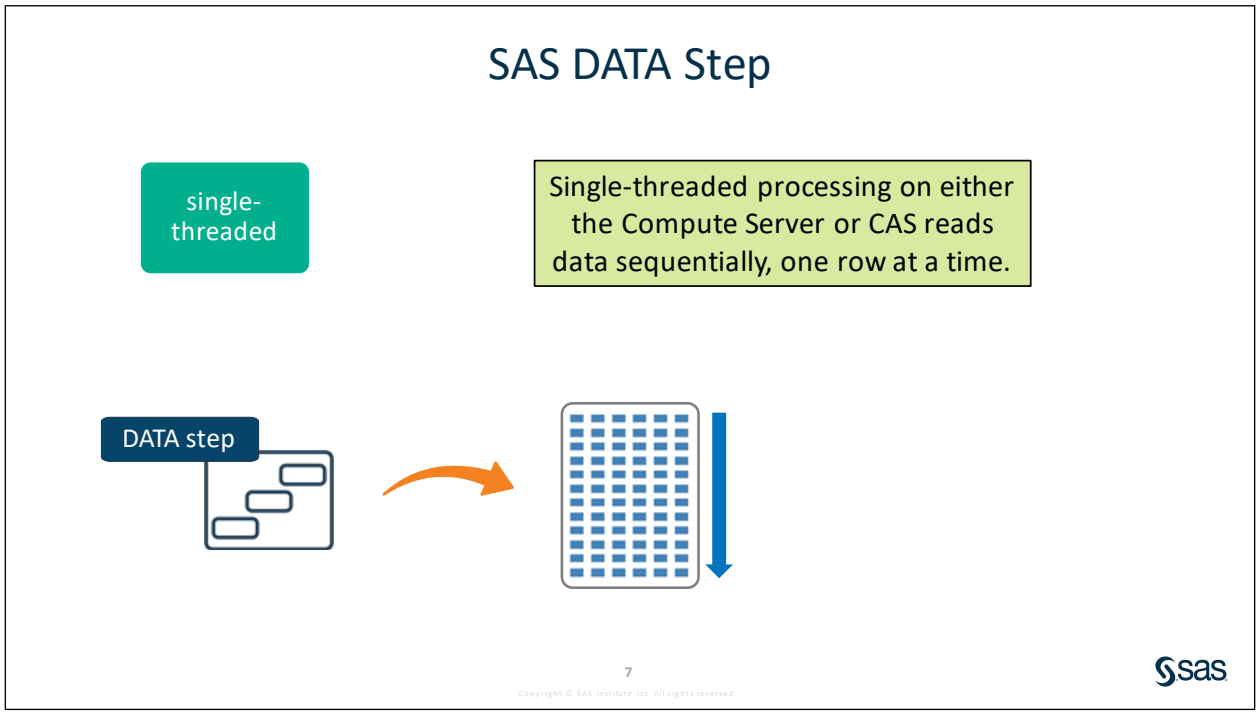
First we focus on the DATA step. You might be wondering whether standard DATA step syntax can execute in CAS to create and manipulate in-memory tables.



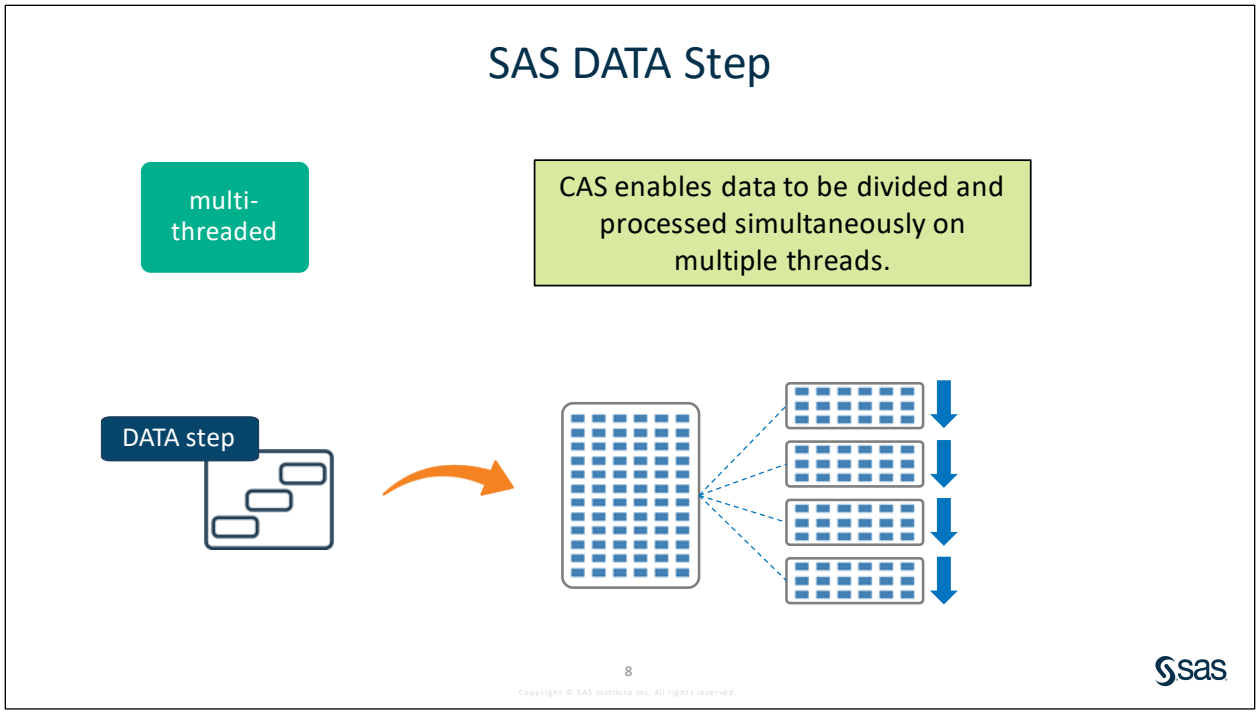
Yes, familiar DATA step syntax can be executed in CAS, but it is important to consider how the data is processed behind the scenes to ensure that you get your desired results.



The DATA step can run on either the SAS Compute Server or CAS. Standard DATA step code that ran successfully in SAS® 9 will run as is on the Compute Server. It will always process data single-threaded. If DATA step code is CAS compliant, you have the option of running the process in CAS either single-threaded or multi-threaded.



When a DATA step runs single-threaded on either the Compute Server or CAS, rows from the input table are read and processed sequentially, one row at a time.

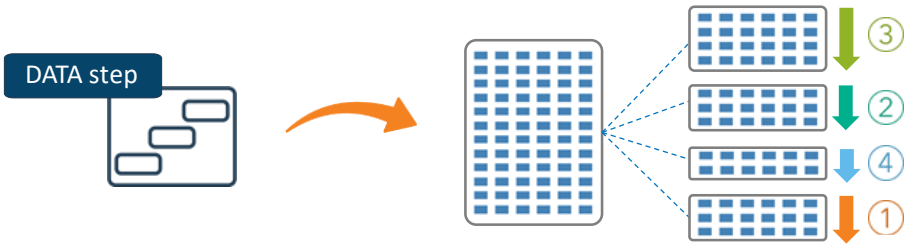


When a DATA step executes as a multi-threaded process, CAS automatically selects the appropriate number of threads to use, distributes the code and data to each thread, and handles the results produced.

SAS DATA Step

multi-threaded

Threads can
- receive different amounts of data
- complete processing and return results to the server controller at different times.



9

The threads might receive different amounts of data, and might complete their processing and return the results in a seemingly random order. CAS reassembles the results. We will look at examples where parallel processing is transparent to the user. The only difference that you will see is faster execution. We will also look at situations where you, as the programmer, need to take additional action to summarize the results from the threads.

continued...

3.01 Activity

Open **pv03a01.sas** from the **activities** folder and perform the following tasks:

1. The first DATA step executes on the SAS Compute Server. The automatic variable **_THREADID_** returns the thread number, and **_NTHREADS_** returns the total number of available threads.
2. Run the first DATA step and view the log. Notice that the DATA step ran on one available thread in the Compute Server.
3. The second DATA step includes the SESSREF= option, specifying that it should run in the **mySession** CAS session. Run this DATA step and view the log. How many threads are available?

10

3.01 Activity

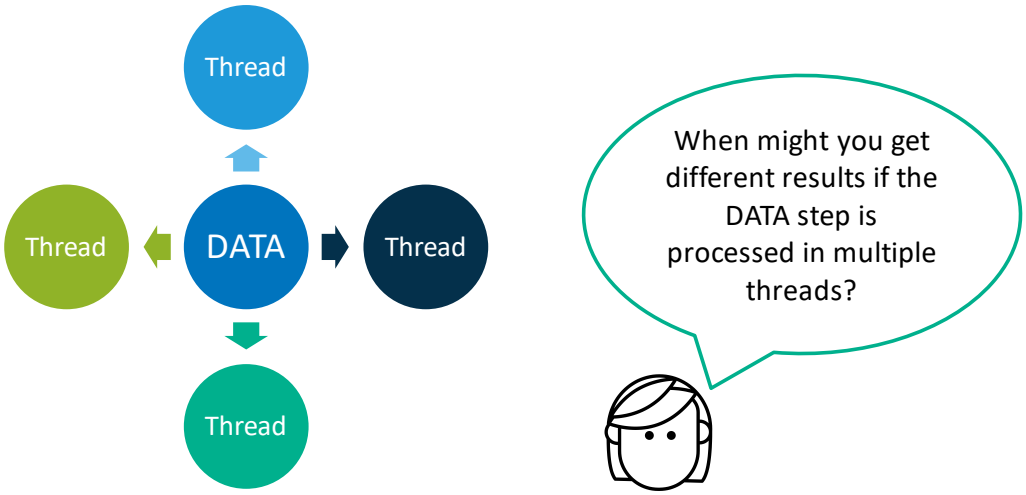
4. Return to the Code tab and modify the code to add the SINGLE=YES option in the DATA statement. Run the step again.

```
data _null_ / sessref="MySession" single=yes;
```

5. Where did the DATA step execute? How many threads were used to process the DATA step?

11

Modifying the DATA Step to Run in CAS



13

Now that we have seen the DATA step run sequentially in a single-threaded Base SAS environment and in the multi-threaded environment in SAS Viya, let's talk about when you might get different results if the DATA step is processed in multiple threads. We will look at changes that we might need to make to our Base SAS DATA step programs to execute in CAS.

Modifying a DATA Step to Run in CAS

Scenario

Modify a DATA step that computes a new column to run in CAS. Explore how the results differ if the step is run single-threaded or multi-threaded.

Program

- **pv03d01.sas**

Syntax

```
PUT _THREADID_= _N_=;
```

Notes

- Use the PROC CASUTIL step to drop the **casuser.orders** table if it already exists as a global-scope table.
- Use the QUIET option in the DROPTABLE statement in PROC CASUTIL to prevent error messages if the table that you are dropping does not exist.
- Use PROC CASUTIL to load a copy of the **pvbase.orders** data set into the global-scope in-memory table, **casuser.orders.**
- The table listed in the DATA statement and the SET statement must reference caslibs in order for the DATA step to process in CAS.
- The END= option is set to 1 when the last row is processed, and the PUT statement writes out to the log the value of **_THREADID_** and the **_N_**.
- When the DATA step runs in CAS, the automatic variable **_N_** returns the number of rows that were processed in the thread (**_THREADID_**= *value*).

Demo

Note: If you have not set up the autoexec file in SAS Studio, open and submit **startup.sas** first.

Open **pv03d01.sas** from the **demos** folder and perform the following tasks:

1. Review the DATA step program that reads **pvbase.orders** and creates the **work.shipping** SAS data set. The DATA step includes regular statements, including WHERE, DROP, IF-THEN, and an assignment statement.
2. Run the DATA step and examine the Log and Output Data tabs. **Work.shipping** includes 235,699 rows and a new computed column, **DaysToDeliver**. Confirm that the values of **Customer_ID** and **Customer_Name** in the first row of the output table are *8818* and *Diab, Ms. Wendy*.

   Note: This DATA step runs on the SAS Compute Server in a single thread. Therefore, the order of the rows in **work.shipping** matches the sequence of the **pvbase.orders** table.

3. In order for this DATA step to run CAS, both the input and output tables must be in-memory tables. Uncomment the PROC CASUTIL step at the top of the program. The DROPTABLE statement drops **casuser.orders** if it exists. The LOAD statement reloads the **orders_hd.sashdat** file into memory as a global-scope table named **casuser.orders**.

   Note: To quickly uncomment the CASUTIL step, highlight the code and press Ctrl+/.

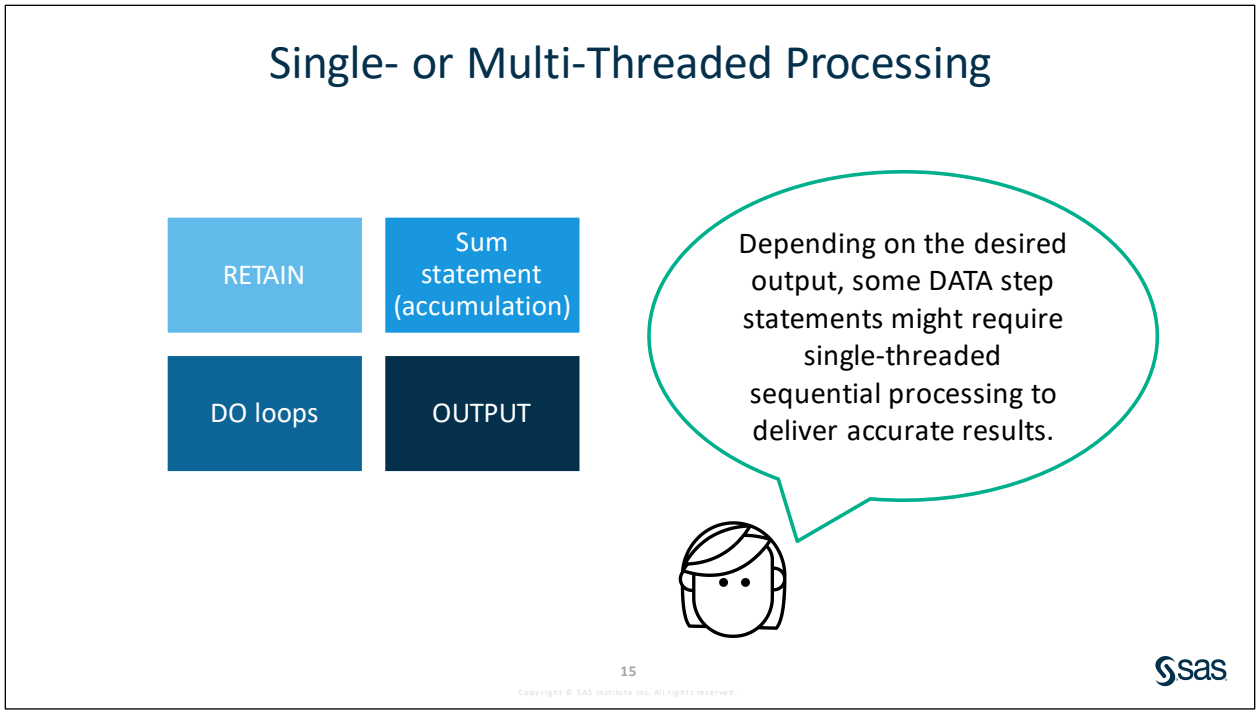
```
proc casutil;
    droptable casdata="orders" incaslib="casuser" quiet;
    load data=pvbase.orders casout="orders" promote;
quit;
```

4. Change the **Work** library reference in the DATA statement to **casuser**. Change the table name **pvbase.orders** in the SET statement to **casuser.orders**.

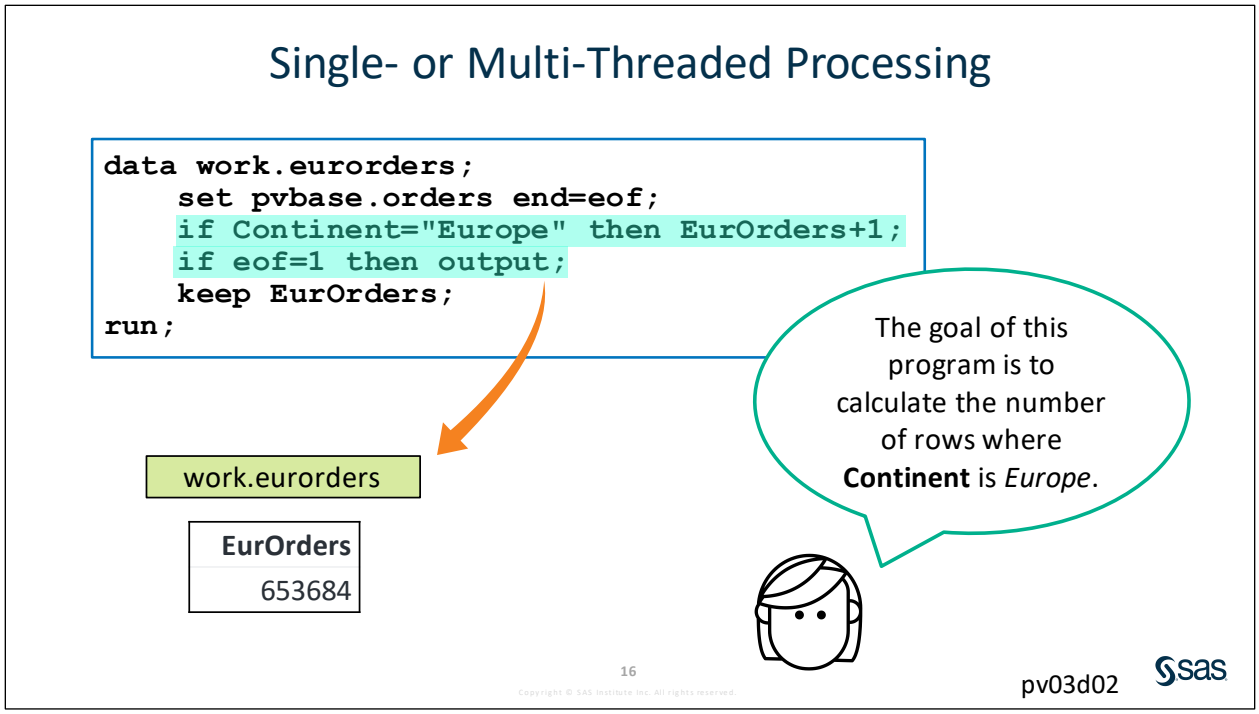
```
data casuser.shipping;
    set casuser.orders end=eof;
...
```

5. Submit the program. Examine the log and confirm that the DATA step ran in CAS. Note the first value of **_THREADID_** listed in the log.
6. Click the **Output Data** tab to view **casuser.shipping**. Notice that the table still includes 235,699 rows. However, the value for **Customer_Name** in the first row is no longer *Diab, Ms. Wendy*.
7. Run the DATA step again and note the first values of **_THREADID_** and **City**. It is likely that both values are different compared to the previous values returned in the DATA step. Worker nodes do not necessarily finish in the same sequence each time that the DATA step runs. Results are returned to the controller in a different order, impacting the order of the output table.

End of Demonstration

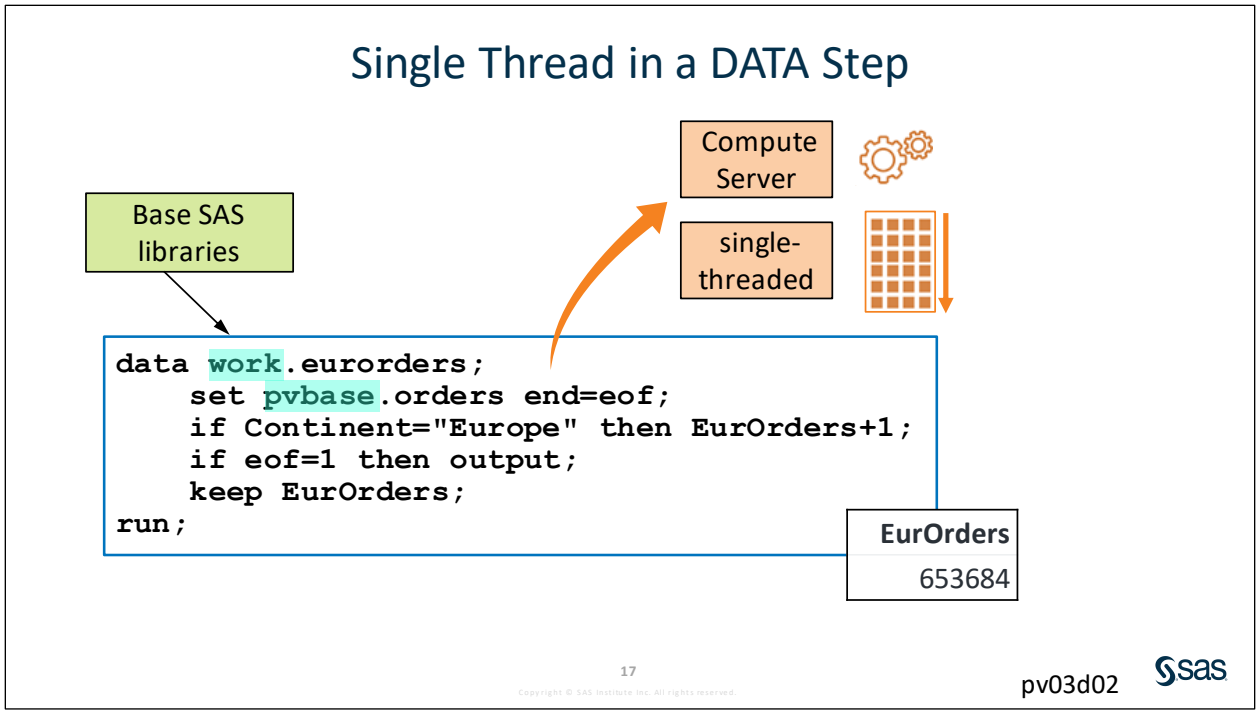


Although many DATA steps might produce the desired output in CAS with no modifications to the code, there are some statements that might require single-threaded processing or another workaround to deliver accurate results.

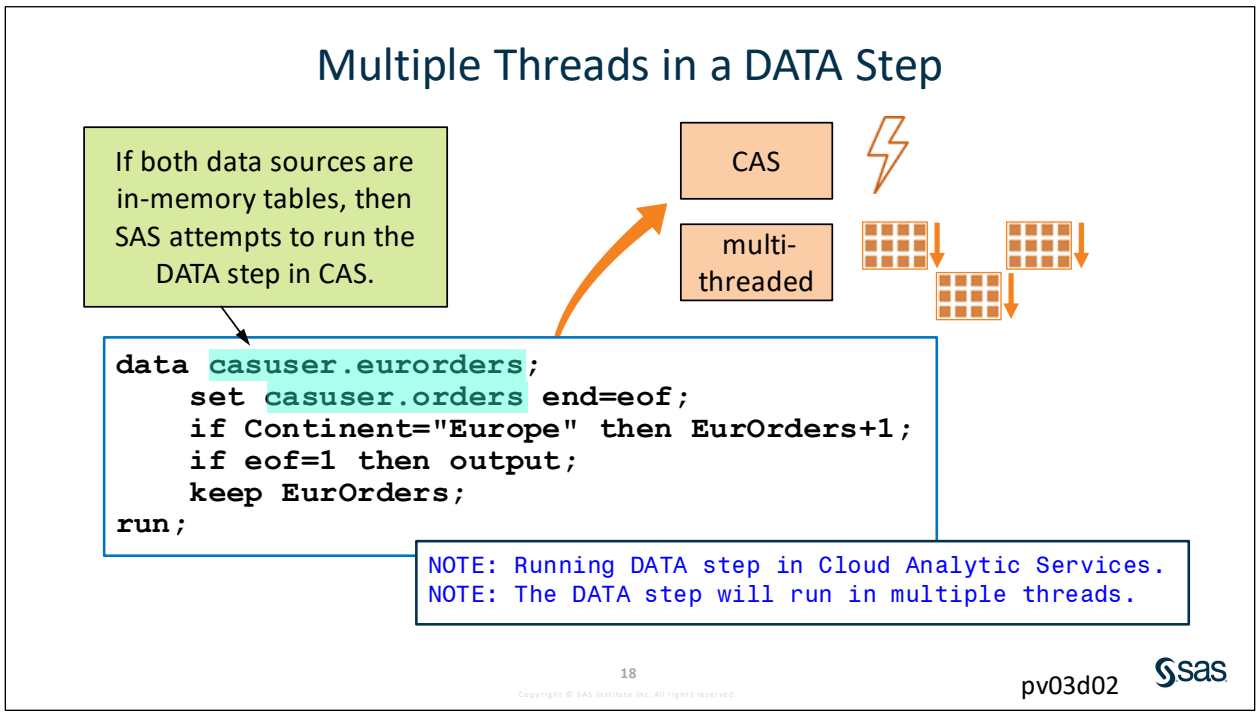


This example uses a sum statement to create an accumulator variable. This DATA step creates the variable **EurOrders** to count the total number of orders from Europe. When we use the sum statement in a DATA step, the expression on the right side of the plus sign is added to the accumulator variable. The value of the accumulator is automatically retained.

Here we are using the sum statement conditionally, so the expression is added to the accumulator only when the condition is true.



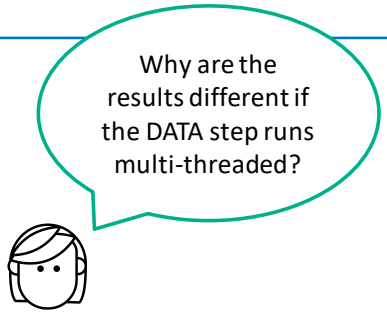
Both the input and output data sets are in SAS libraries, so this code is processed sequentially in a single thread. One row is output with the value for the total number of European orders.



What about this version of the program? Both the input table, **orders**, and the output table, **eurorders**, are in the **Casuser** caslib, so the step executes in CAS across multiple threads.

Multiple Threads in a DATA Step

```
data casuser.eurorders;
    set casuser.orders end=eof;
    if Continent="Europe"
       then EurOrders+1;
    if eof=1 then output;
    keep EurOrders;
run;
```

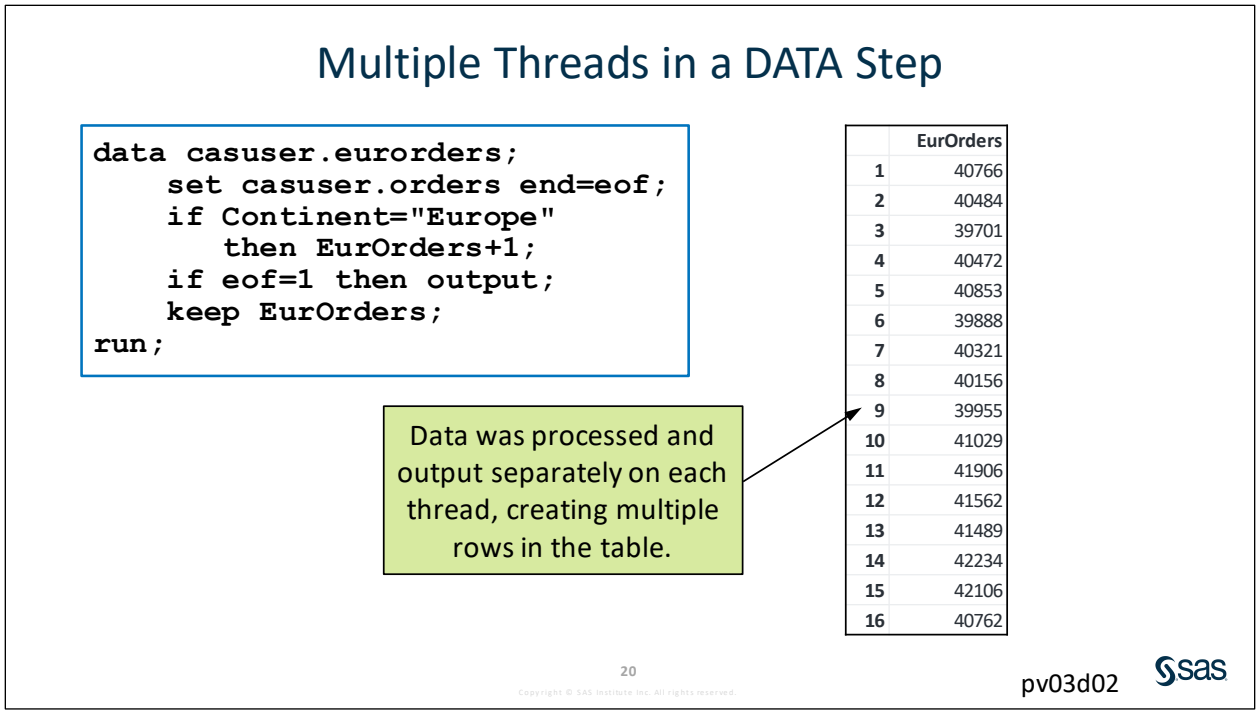


	EurOrders
1	40766
2	40484
3	39701
4	40472
5	40853
6	39888
7	40321
8	40156
9	39955
10	41029
11	41906
12	41562
13	41489
14	42234
15	42106
16	40762

19

pv03d02

But we do not get one row. We get 16 rows! So exactly what is going on here?



When steps are processed in CAS, data is distributed across multiple threads, and the step is executed on each thread simultaneously. After processing its rows, each thread or machine returns the count of **EurOrders** for the data it received. The controller reassembles the results by appending the rows, not summing them.

This divide and conquer approach can perform the aggregation faster because the work is distributed. But, as you can see, we do not get the same results as our Base SAS DATA step. What can we do to generate accurate results?

This is a case where you need to take additional action to summarize the results from the threads. You can write a simple DATA step to sum the values of the multiple rows, to get the total number of European orders.

3.02 Activity

Open **pv03a02.sas** from the **activities** folder and perform the following tasks:

1. Run the program and view the output data. Examine the log and note the real time and CPU time it takes to execute the code in multiple threads.
2. Add the SINGLE=YES option to run the DATA step in CAS in a single thread.

```
data casuser.eurorders / single=yes;
```

3. Run the program. Which DATA step had the shortest real time for processing?

21

Summarizing in CAS with the Sum Statement

Scenario

Execute two DATA steps in CAS to get the same results as a DATA step executed on the Compute Server.

Program

- **pv03d03.sas**

Syntax

```
DATA caslib.CAS-table-name | <SINGLE=YES>;
     SET caslib.CAS-table-name;
...
RUN;
```

Notes

- The CAS table **casuser.orders** has already been loaded into CAS and is a copy of the **pvbase.orders** data set.
- The sum statement creates an accumulator variable.
- The DATA step runs in CAS when the input table and the output table reference CAS in-memory tables.
- The goal is to have CAS process the data as quickly as possible while producing the same results as the original Base SAS DATA step.

Demo

Open **pv03d03.sas** from the **demos** folder and perform the following tasks:

Note: If you have not set up the autoexec file in SAS Studio, open and submit **startup.sas** first. If **orders** has not been loaded into memory, uncomment and run the CASUTIL step.

1. The first DATA step runs on the Compute Server and processes the data in a single thread. It correctly calculates the number of orders placed in Europe and writes the value to the output table. Highlight and run the first DATA step. Examine the log and note the values for real time and cpu time.

   Note: Your time values might differ.

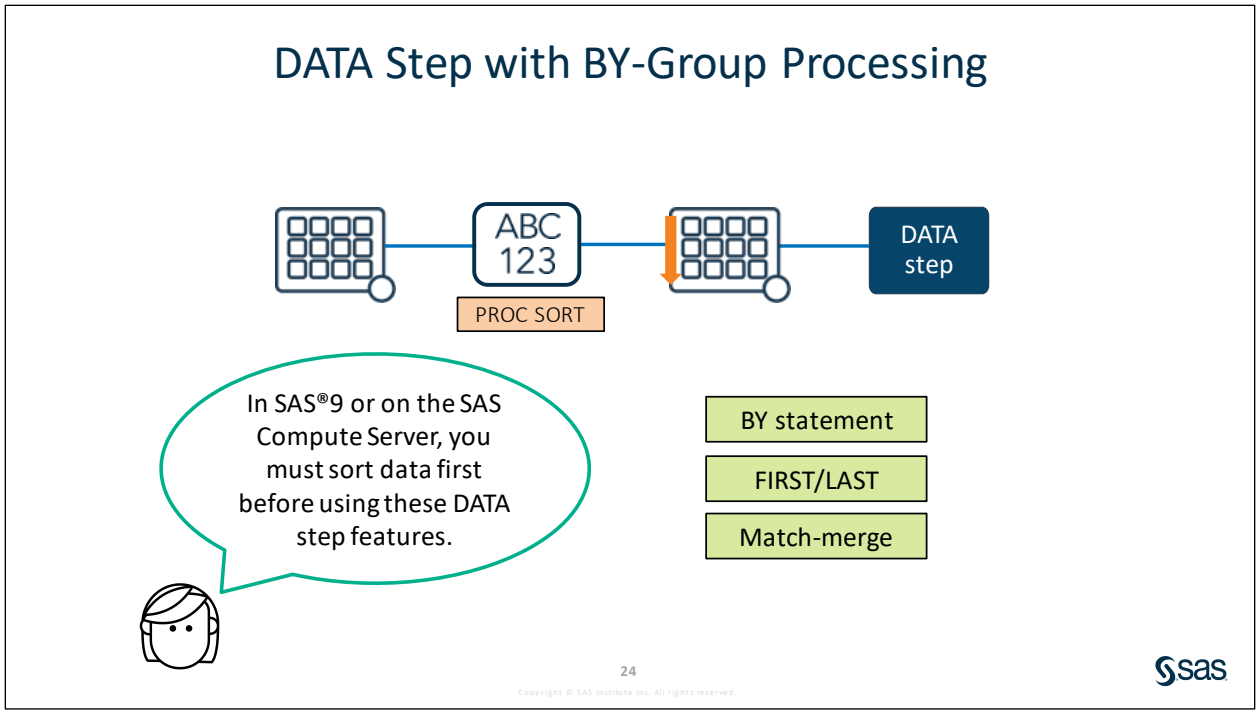
```
NOTE: DATA statement used (Total process time):
      real time           4.48 seconds
      cpu time            1.08 seconds
```

2. The alternate solution to processing all the rows sequentially in the Compute Server is to use two DATA steps processed in CAS. The first DATA step uses multiple threads and creates **casuser.eurorders_thread** with multiple rows (one from each thread). CAS returns an accumulating variable, **EurOrders_thread**, from each thread. The second DATA step sums the multiple rows from **casuser.eurorders** into one row using a single thread.

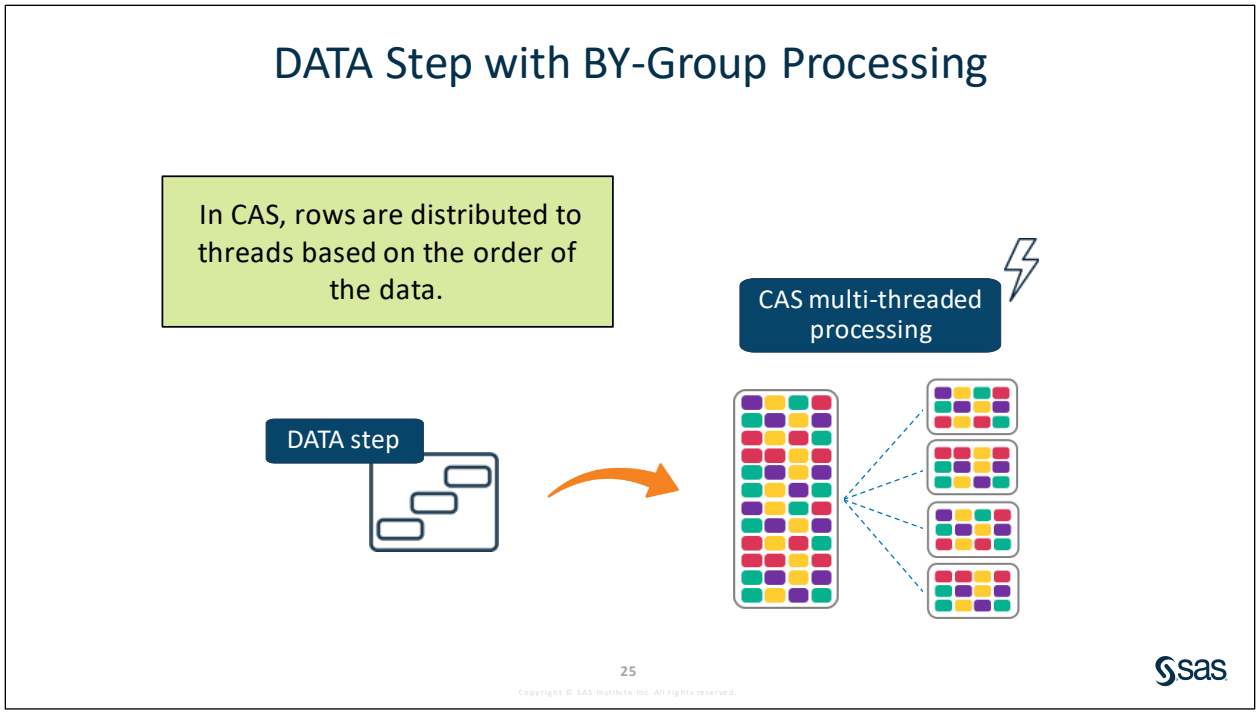
3. Highlight and run the two DATA steps that will run in CAS. View the log and note the values for real time and cpu time. The combined real time for the two DATA steps was faster than processing all the rows in a single thread. Although processing times can vary from run to run, the relationship between the runs should remain the same, so we can conclude that the two-DATA-step process is faster than using one single-threaded DATA step.

```
82   /* DATA step 1 */
83   data casuser.eurorders_thread;
84       set casuser.orders end=eof;
85       if Continent="Europe" then EurOrders_thread+1;
86       if eof=1 then output;
87       keep EurOrders_thread;
88   run;
NOTE: Running DATA step in Cloud Analytic Services.
NOTE: The DATA step will run in multiple threads.
NOTE: There were 951669 observations read from the table ORDERS in caslib CASUSER(student).
NOTE: The table eurorders_thread in caslib CASUSER(student) has 16 observations and 1 variables.
NOTE: DATA statement used (Total process time):
      real time           0.10 seconds
      cpu time            0.00 seconds

89
90   /* DATA Step 2 */
91   data casuser.eurorders / single=yes;
92       set casuser.eurorders_thread end=eof;
93       EurOrders+EurOrders_thread;
94       keep EurOrders;
95       if eof then output;
96   run;
NOTE: Running DATA step in Cloud Analytic Services.
NOTE: There were 16 observations read from the table EURORDERS_THREAD in caslib CASUSER(student).
NOTE: The table eurorders_total in caslib CASUSER(student) has 1 observations and 1 variables.
NOTE: DATA statement used (Total process time):
      real time           0.01 seconds
      cpu time            0.01 seconds
```

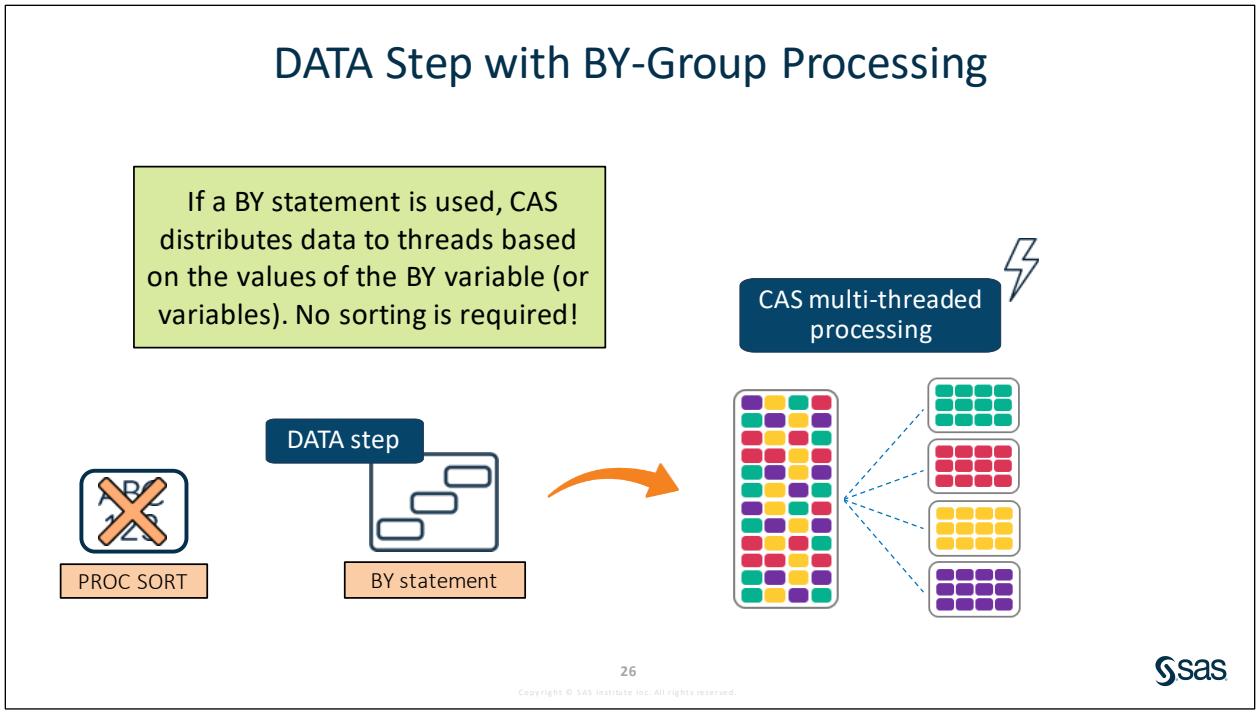
End of Demonstration



The BY statement in the DATA step is a powerful way to process data in groups. The BY statement is used in combination with FIRST and LAST variables to identify the first and last row within each group. The BY statement is also used when merging tables to identify matching rows. If a BY statement is used in a DATA step that is processed in SAS®9 or the SAS Compute Server, the data set must be sorted first. Sorting can be resource intensive, especially with very large data sets. In addition, when the DATA step is processed in SAS®9 or the Compute Server, the rows are processed sequentially in a single thread.

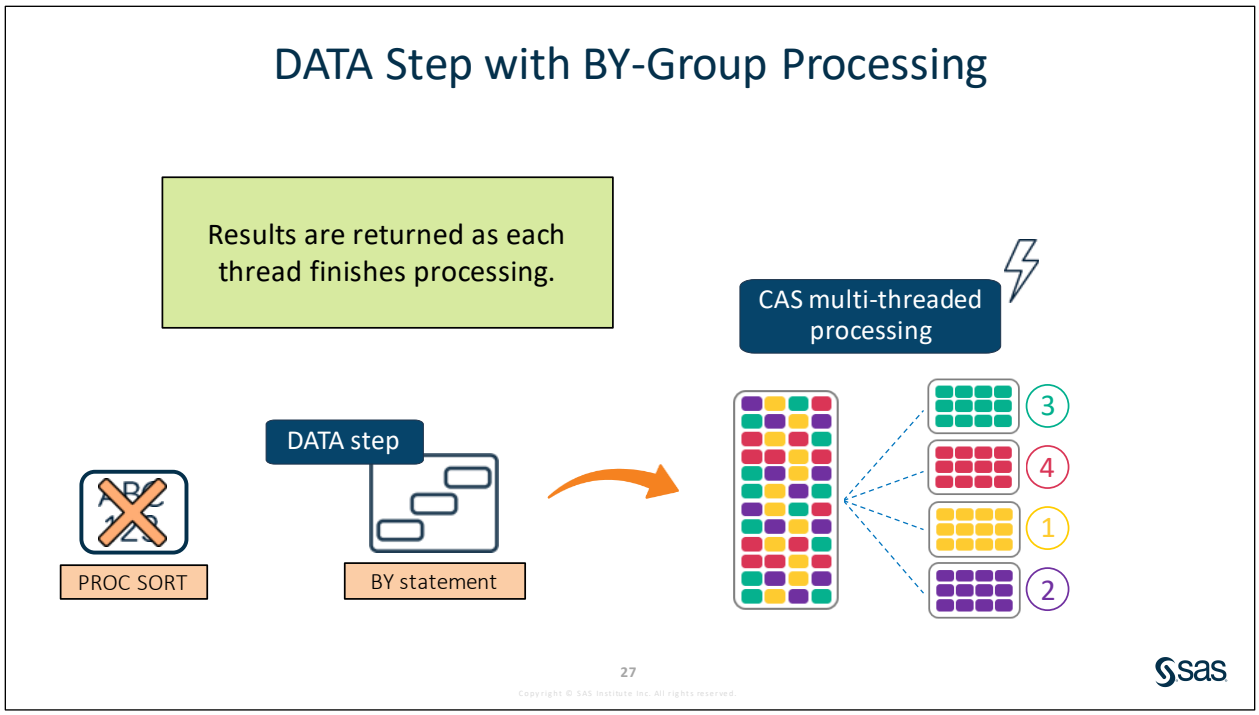


The default behavior when data is loaded into CAS is to distribute the input data based on the original order among the different threads or multiple machines. The DATA step is executed among the different threads or on multiple machines.



When we add a BY statement to the DATA step processed in CAS, the rows are grouped based on the first BY variable and then distributed across multiple threads or machines. Rows within the same BY group are processed by the same thread or worker. Because the data is automatically distributed based on the value of the BY variable, PROC SORT steps are no longer necessary.

If you apply a format to the BY variable, the rows are distributed based on the formatted values.



The DATA step with the BY statement executes on each thread, and results are returned as each thread finishes processing. The order might be different each time that the program executes.

BY-Group Processing in CAS

Scenario

Modify a program to run a DATA step with a BY statement in CAS.

Program

- **pv03d04.sas**

Syntax

```
DATA caslib.CAS-table-name;
     SET caslib.CAS-table-name;
     BY BY-variable;
 …
RUN;
```

Notes

- When a BY statement is used in a DATA step that processes on the SAS Compute Server, the input table must be sorted.
- When a BY statement is used in a DATA step that processes in CAS, the input table does not need to be sorted. CAS automatically distributes the data to the threads based on the values of the BY variable (or variables).

Demo

Open **pv03d04.sas** from the **demos** folder and perform the following tasks:

Note: If you have not set up the autoexec file in SAS Studio, open and submit **startup.sas** first. If **orders** has not been loaded into memory, uncomment and run the CASUTIL step.

1. This program first sorts the **pvbase.orders** table by **Continent**. The DATA step includes a BY statement to process the data in groups. **TotalCost** represents the sum of **Cost** for each value of **Continent**.

```
proc sort data=pvbase.orders out=orders_s;
    by Continent;
run;

data work.ContTotals;
    set orders_s;
    by Continent;
    if first.Continent then TotalCost=0;
    TotalCost+Cost;
    if last.Continent then output;
    keep Continent TotalCost;
    format TotalCost dollar15.2;
run;
```

2. Submit the program and examine the Output Data tab for **work.ContTotals**. The values are in sorted order by **Continent**.

   Note: In SAS Studio, the **work.orders_s** table appears on the Output Data tab by default. Click the drop-down arrow to select the **work.ContTotals** table.

	Continent	TotalCost
1	Africa	$87,096.20
2	Asia	$125,983.80
3	Europe	$50,600,011.32
4	North America	$18,518,294.90
5	Oceania	$4,666,493.04

3. Modify the program to run in CAS.

   a. Delete the PROC SORT step.

   b. Change **work.ContTotals** to **casuser.ContTotals** in the DATA statement.

   c. Change **orders_s** to **casuser.orders** in the SET statement.

```
data casuser.ContTotals;
    set casuser.orders;
    by Continent;
    if first.Continent then TotalCost=0;
    TotalCost+Cost;
    if last.Continent then output;
    keep Continent TotalCost;
    format TotalCost dollar15.2;
run;
```

4. Submit the program and examine the Output Data tab for **casuser.ContTotals**. The results for **TotalCost** are the same, but the rows are not in sorted order by **Continent**. This is because the rows are returned to the controller in the order the threads finish processing.

   Note: To view the results in sorted order, right-click **Continent** and select **Sort** ⇨ **Ascending**. This action does not permanently sort the **casuser.ContTotals** table, but it arranges the rows for viewing.

	Continent	TotalCost
1	Africa	$87,096.20
2	North America	$18,518,294.90
3	Europe	$50,600,011.32
4	Oceania	$4,666,493.04
5	Asia	$125,983.80

End of Demonstration

3.03 Activity

Open **pv03a03.sas** from the **activities** folder and perform the following tasks:

1. Submit the program and examine the values of **_NTHREADS_** , **_THREADID_**, and **_N_** in the log.
2. Were all of the available threads used in CAS to process this DATA step?

29

DATA Step Restrictions in CAS

SAS Cloud Analytic Services (CAS)

```
data casuser.retail;
    set casuser.orders;
    by Continent descending RetailPrice;
...
run;
```

CAS allows the DESCENDING keyword on all variables except for the first one listed in the BY statement.

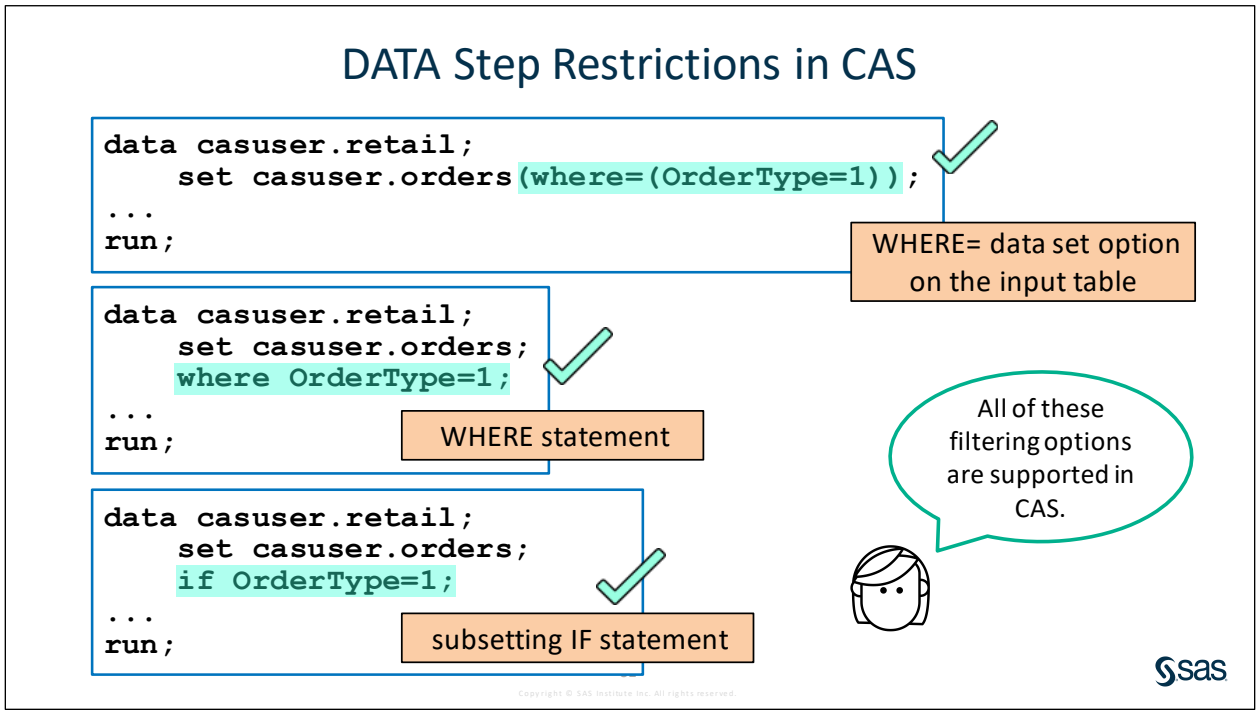
```
data casuser.retail;
    set casuser.orders;
    by descending RetailPrice;
...
run;
```

SAS Compute Server

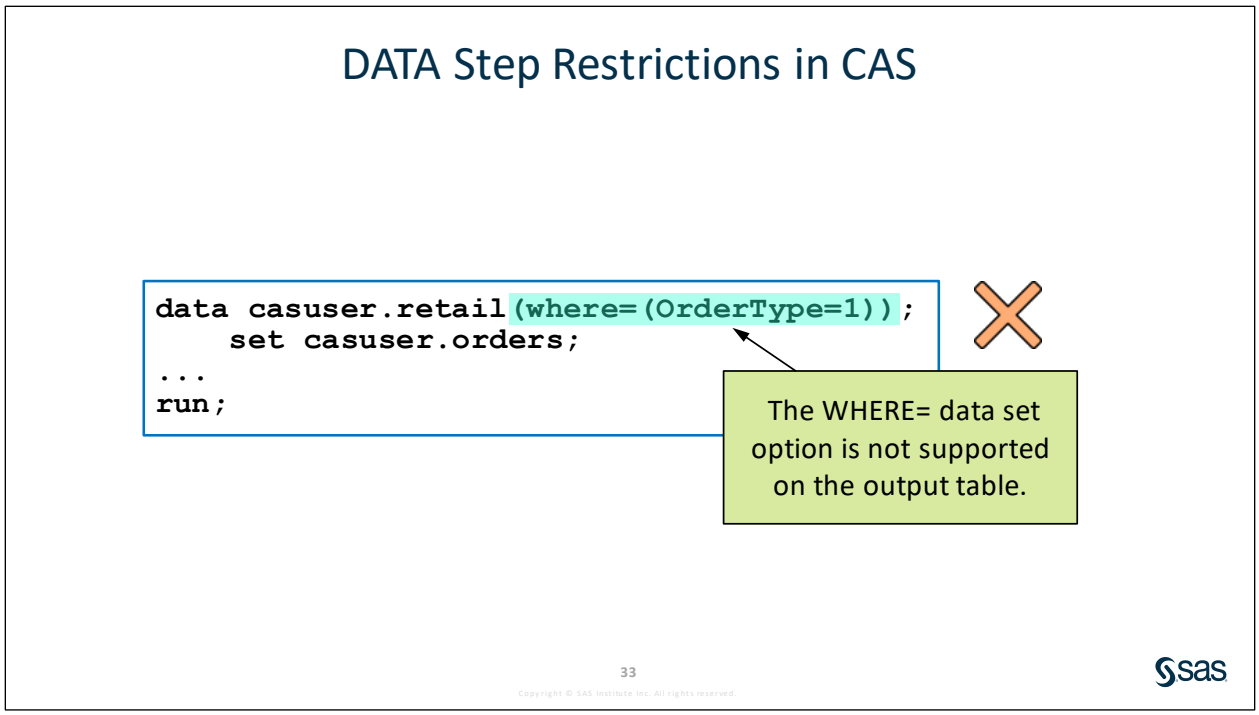
31

Starting with SAS Viya 3.5, CAS supports the DESCENDING option in the BY statement as long as it is not used on the first variable listed. Recall that the first variable in the BY statement controls how data is grouped across the threads, but it does not order the values. Threads return results to the controller in the order in which they complete processing rather than in a sorted sequence. After data is grouped by the first variable on separate threads, each thread can sort the subset of rows by either ascending or descending sequence.

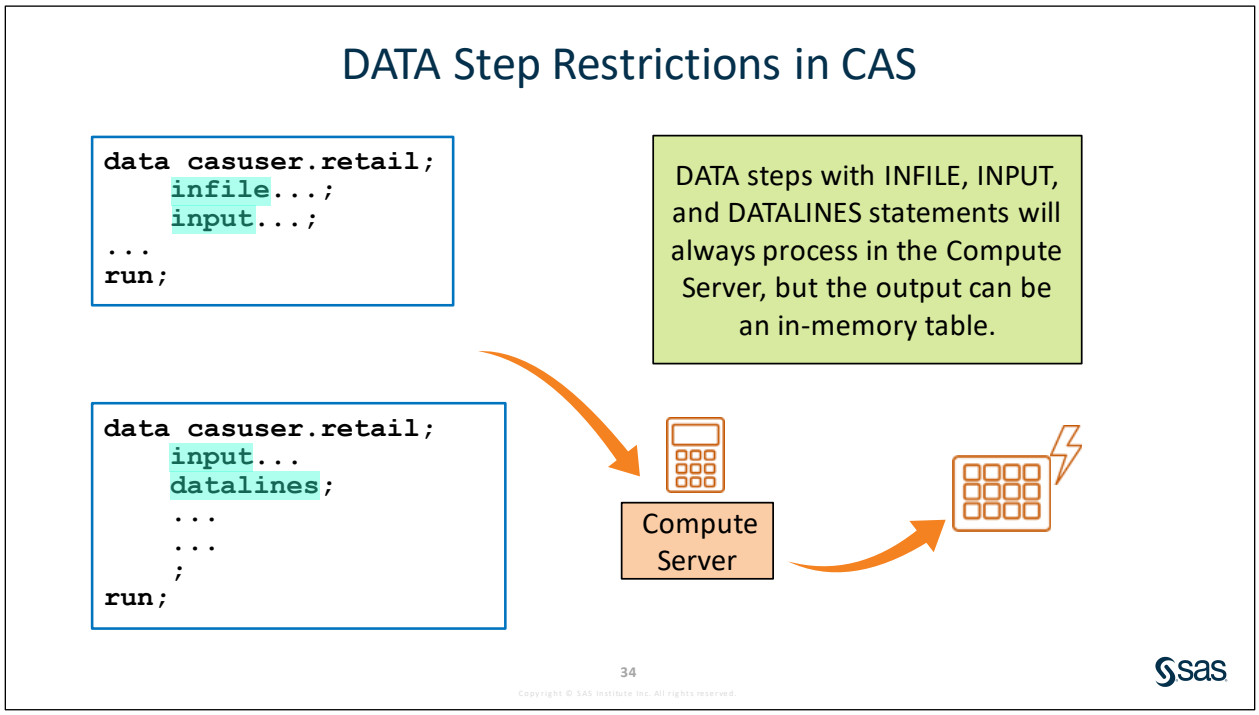
If the DESCENDING option is used on the first BY variable, the step processes on the Compute Server.



The WHERE= data set option in the SET statement or the WHERE statement can be used in CAS to filter the input data. To read all rows from the input table and then filter the rows written to the output table based on a condition, use the subsetting IF statement.



CAS does not support the use of the WHERE= data set option on an output table. If you have any programs that are currently using the WHERE= data set option in the DATA statement, you must delete this option.

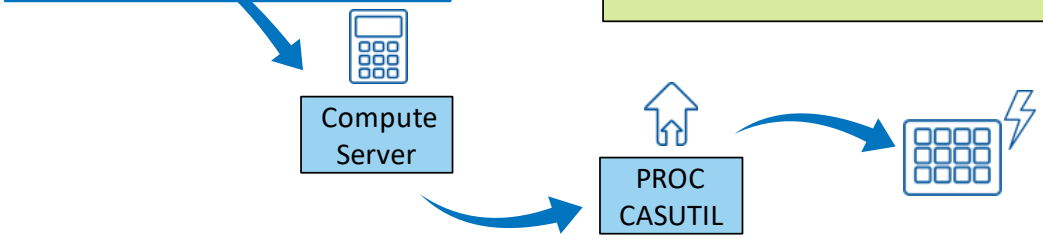


The INFILE, INPUT, and DATALINES statements are supported only on the Compute Server. You can use these statements to create an in-memory table in CAS from raw data files or embedded data. However, the DATA step will execute in the Compute Server and transfer the data into CAS.

DATA Step Restrictions in CAS

```
data pvbase.retail;
     modify pvbase.orders;
     replace...;
     remove...;
...
run;
```

MODIFY, REMOVE, and REPLACE statements cannot update in-memory tables in CAS. Update the SAS data set on the Compute Server and then load to memory.



35

The MODIFY, REMOVE, and REPLACE statements are used in the DATA step to update data in place, so no extra copy of the data is created. SAS Viya does not allow updating rows of an in-memory table, so these statements are not supported in a DATA step that is running in CAS. If you have SAS programs that use these statements, you can continue to use them and run the code on the Compute Server. After the Base SAS table has been updated, you can load the table into memory in CAS.

DATA Step Restrictions in CAS

SAS Functions and CALL Routines by Category

CAS	ABS Function	Returns the absolute value.
	AIRY Function	Returns the value of the Airy function.
	ANYALNUM Function	Searches a character string for an alphanumeric character, and returns the first position at which the character is found.
	ANYALPHA Function	Searches a character string for an alphabetic character, and returns the first position at which the character is found.
	ANYCNTRL Function	Searches a character string for a control character, and returns the first position at which that character is found.
	ANYDIGIT Function	Searches a character string for a digit, and returns the first position at which the dig

Some functions run in SAS only, and some functions run in SAS and CAS.

Check SAS documentation for a full list of functions supported in CAS.

36

DATA Step Restrictions in CAS

Data Value	Compute Server Format	Formatted Value	CAS Alternative
22036	WORDDATE.	May 1, 2020	NLDATE.
2.5	WORDS30.	two and fifty hundredths	none
2.5	WORDF20.	two and 50/100	none

These formats are not supported in CAS.

37

The WORDDATE format is not supported in CAS. If you use WORDDATE to display dates in your programs, you need to modify your code to use the SAS National Language Support (NLS) equivalent format NLDATE.

The WORDS and WORDF formats are not supported in a DATA step that is running in CAS. If you use the WORDS and WORDF format in your program, you need to remove them. There are no alternative CAS formats.

To see a full list of supported formats in CAS, view **SAS 9.4 Formats and Informats: Reference** and search for **Formats by Category**.

DATA Step Restrictions in CAS

default notes

```
OPTIONS MSGLEVEL=N;
```

additional notes

```
OPTIONS MSGLEVEL=I;
```

MSGLEVEL=I can provide more detailed messages in the log regarding CAS processing.

```
data casuser.retail(where=(OrderType=1));
    set casuser.orders;
run;
```

```
NOTE: The WHERE data set option on the DATA statement is not
      supported with DATA step in Cloud Analytic Services.
```

38

The system option MSGLEVEL enables you to control the level of detail in log messages. MSGLEVEL=N provides default notes, but if you set MSGLEVEL=I, then additional notes might provide insight into how your code is processing. For example, the additional note regarding this DATA step indicates the reason that it cannot run in CAS is because the WHERE data set option is not supported in the DATA statement.

3.04 Activity

Open **pv03a04.sas** from the **activities** folder and perform the following tasks:

1. Run the program and view the log. The log messages indicate that the step ran in the Compute Server.
2. Add OPTIONS statements to set the MSGLEVEL= option to **I** at the start of the program and **N** at the end of the program.
3. Run the program and read the notes in the log. Which portion of the DATA step ran in CAS? Why did the entire DATA step not process in CAS?
4. Modify the DATA step to run in CAS and resubmit the step. Confirm in the log that the entire step processes in CAS.

39

Practice

If your CAS session has timed out or has been terminated, or SAS Studio timed out, open and submit **startup.sas** before starting the practice.

Level 1

1. Modifying a DATA Step to Run in CAS (Creating New Variables)

a. Open **pv03p01.sas** from the **practices** folder and submit the program. Examine the log.

Note: If **orders** has not been loaded into memory, uncomment and run the CASUTIL step.

How long did it take to process the code in Base SAS?

- Real Time: _____
- CPU Time: _____

b. Modify the DATA step to run in CAS. Make the following changes:

1) Change **work.CouponMailer** to **casuser.CouponMailer** in the DATA statement.
2) Change **pvbase.orders** to **casuser.orders** in the SET statement.
3) Change **worddate.** to **nldate.** in the FORMAT statement.

c. Submit the modified program and examine the log.

How long did it take to process the DATA step in Base SAS?

- Real Time: _____
- CPU Time: _____

Level 2

2. Modifying a DATA Step to Run in CAS (Accumulating Totals)

a. Open **pv03p02.sas** from the **practices** folder. Run the program and examine the results.

Note: If **orders** has not been loaded into memory, uncomment and run the CASUTIL step.

Orion Star Club Membership Counts

Obs	Total # Gold Memberships	Total # Internet/Catalog Memberships	Total # Other Memberships
1	483438	76965	391266

b. Optimize the program to run in CAS by using two DATA steps. Add a FORMAT statement to display commas for the all variables.

c. Submit the modified program and examine the results. Confirm that the numbers match the output created from the Compute Server.

Challenge

3. Modifying a DATA Step to Run in CAS (BY-Group Processing)

a. Open **pv03p03.sas** from the **practices** folder.

Note: If **orders** has not been loaded into memory, uncomment and run the CASUTIL step.

b. Modify the DATA step to run in CAS. Submit the step and verify that the same number of rows (10,779) were created in the **CityTotals** table.

1) Are the results from the Base SAS DATA step and the results from the DATA step that ran in CAS identical?

2) Can you tell by the results how many threads were used to process the data in CAS?

c. Modify the DATA step to use _THREADID_= and _ NTHREADS _= to add a message in the log showing how many threads were used to process the DATA step in CAS and how many total threads are available on the CAS server. Run the DATA step.

1) How many threads were used to process the DATA step that ran in CAS?

2) Were all available CAS threads used for this process?

d. Uncomment the remainder of the program and submit it. Verify that the calculations for **work.CityTotals** and **casuser.CityTotals** have the same values for *Abbeville, North America*.

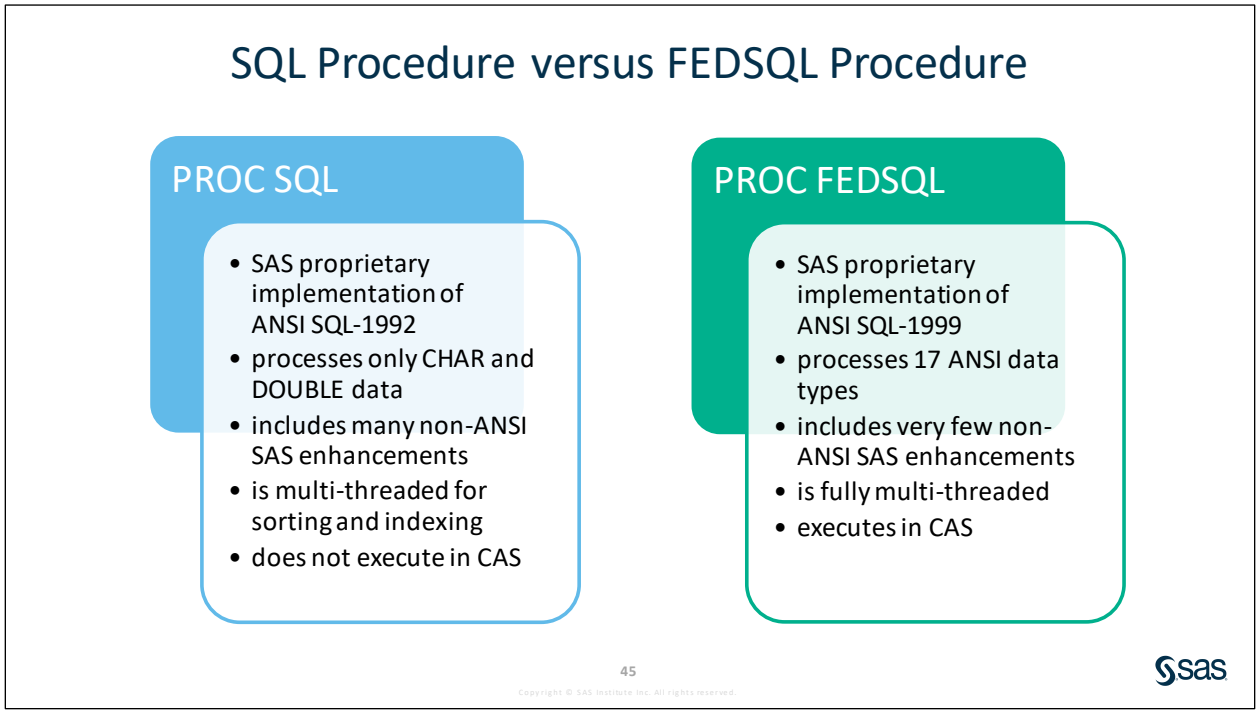
e. Examine the results and the log.

1) Examine the log. Did CAS process any portion of the second PROC PRINT step?

2) Did both processes produce the same value for *Abbeville, North America*?

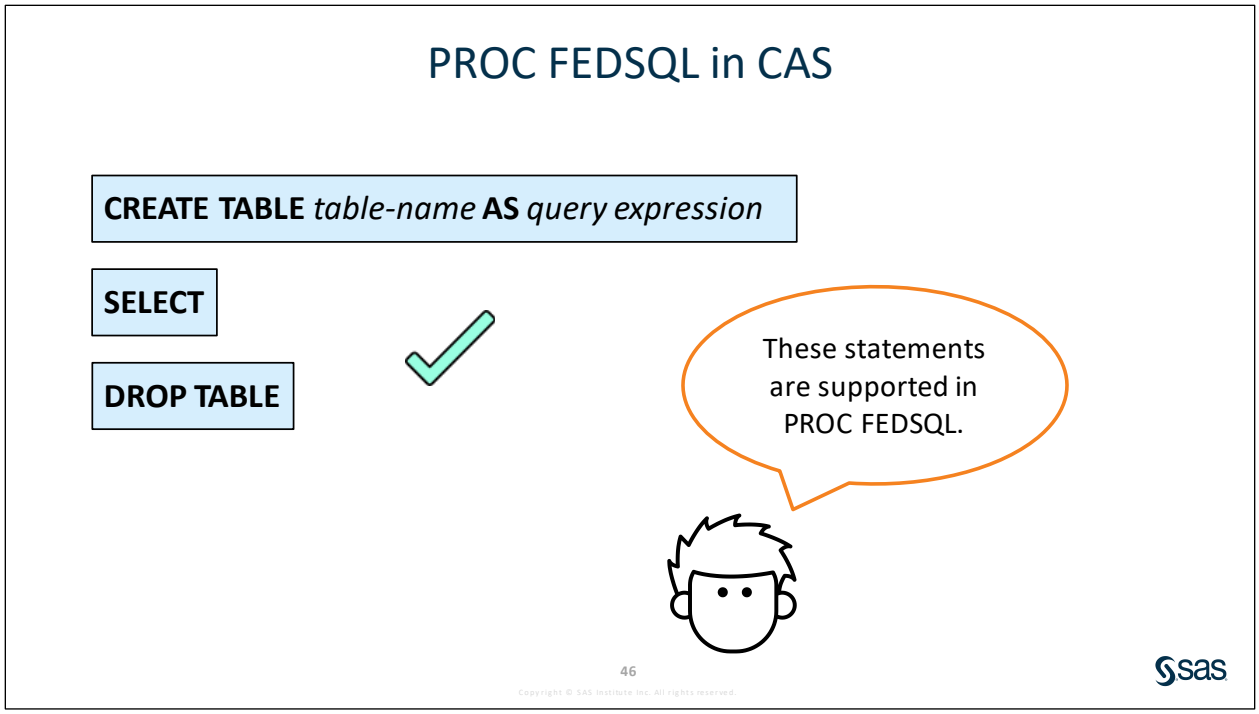
End of Practices

3.2 Modifying SQL Code for CAS



PROC SQL is a single-threaded procedure that can run only on the Compute Server. FedSQL is a SAS proprietary implementation of ANSI SQL-1999 and is invoked in SAS using PROC FEDSQL. FedSQL extends its CAS data type support to include INT64 and INT32 data types in addition to CHAR, DOUBLE, and VARCHAR. The other major advantage PROC FEDSQL has over the traditional SQL procedure is that FEDSQL can process natively in CAS.

FedSQL Data Type	CAS Data Type	Description
BIGINT	INT64	Large signed, exact whole number.
CHAR(*n*)	CHAR	Stores a fixed-length character string, where *n* is the maximum number of characters to store. The maximum number of characters is required to store each value regardless of the actual size of the value. If CHAR(10) is specified and the character string is only five characters long, the value is right-padded with spaces.
DOUBLE	DOUBLE	Stores a signed, approximate, double-precision, floating-point number. Allows numbers of large magnitude and permits computations that require many digits of precision to the right of the decimal point. For the CAS server, this is a 64-bit double-precision, floating-point number.
INTEGER	INT32	Regular signed, exact whole number.
VARCHAR(n)	VARCHAR(*n*)	Stores a varying-length character string.



You will find that PROC FEDSQL has similar syntax to PROC SQL. However, not all statements and functionality are available in PROC FEDSQL. The CREATE TABLE statement that creates a table from the results of a query expression is supported, the DROP TABLE statement is available, and the SELECT statement is available in PROC FEDSQL.

Executing SQL Queries in CAS Using PROC FEDSQL

Scenario

Modify a PROC SQL query to run in CAS by converting it to PROC FEDSQL.

Program

- **pv03d05.sas**

Syntax

```
PROC FEDSQL SESSREF=cas-session-name;
SELECT col-name, col-name
      FROM CAS-table
      WHERE expression
      GROUP BY item-name
      HAVING col-name
      ORDER BY item-name;
QUIT;
```

Notes

- If the SQL query is a simple query, you can make minimal changes to it to get it to run in CAS.
- For all FedSQL queries to run in CAS, you have to use the SESSREF= option to specify the CAS session name, and your table listed in the FROM clause must be a table that is loaded to an in-memory table in CAS.

Demo

Open **pv03d05.sas** from the **demos** folder and perform the following tasks:

Note: If you have not set up the autoexec file in SAS Studio, open and submit **startup.sas** first. If **orders** has not been loaded into memory, uncomment and run the CASUTIL step.

1. The program includes a PROC SQL step that will run on the Compute Server. Submit the program to review the results.

Customer Name	City Name
Adams, Mr. Johann	
Mahomed, Mr. Neil	
Venter, Ms. Sunita	
Bacus, Ms. Hilary	Arcadia
Lambert, Ms. Raeesa	Arcadia
Loubser, Ms. Marlene	Arcadia

2. Make a copy of the original program (highlight the program and press Ctrl+C) and paste it (press Ctrl+V) below the original PROC SQL step.
3. Make the following modifications:
   a. Change **sql** to **fedsql** in the PROC statement.
   b. Add **sessref=mysession** to the PROC FEDSQL statement.

c. Change the **pvbase.orders** name in the FROM clause to **casuser.orders**.

4. Run the PROC FEDSQL step and examine the log. Note that there is no clear message indicating that the step was processed in CAS.

```
...
proc fedsql sessref=mysession;
select distinct Customer_Name, City
    from casuser.orders
    where Continent='Africa'
    order by City;
quit;
```

5. Add an OPTIONS statement before the PROC FEDSQL step to set the MSGLEVEL= option to I. This provides more detailed notes in the log. Reset the MSGLEVEL= options to N at the end of the program. Submit the modified program and view the log. The notes indicate that the query was processed in CAS.

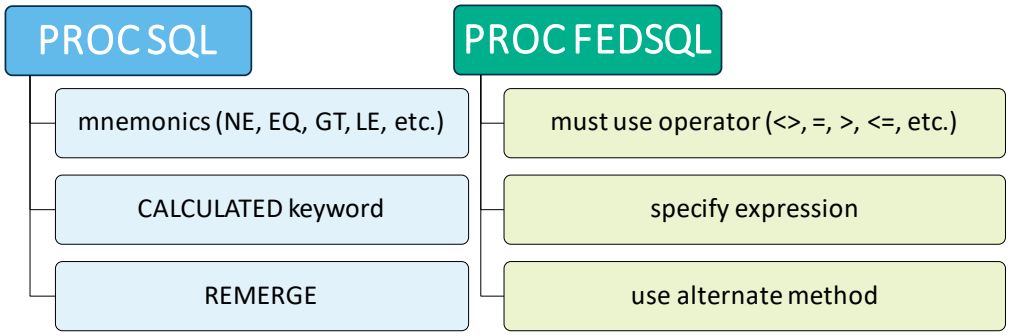
```
83   proc fedsql sessref=mysession;
NOTE: FEDSQL: Running on CAS due to "sessref".
```

6. Review the results to make sure that the results match the original results from the PROC SQL step.

Customer Name	City Name
Adams, Mr. Johann	
Mahomed, Mr. Neil	
Venter, Ms. Sunita	
Bacus, Ms. Hilary	Arcadia
Lambert, Ms. Raeesa	Arcadia
Loubser, Ms. Marlene	Arcadia

End of Demonstration

PROC SQL versus PROC FEDSQL



48

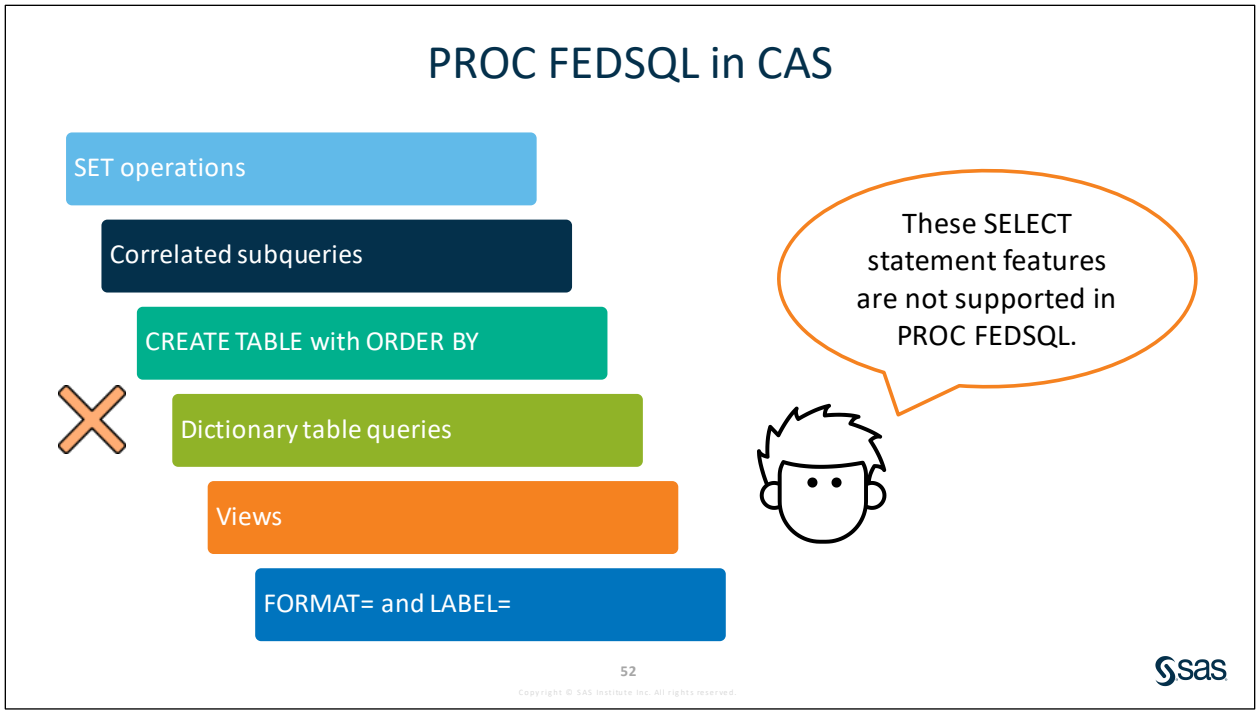
3.05 Activity

Open **pv03a05.sas** from the **activities** folder and perform the following tasks:

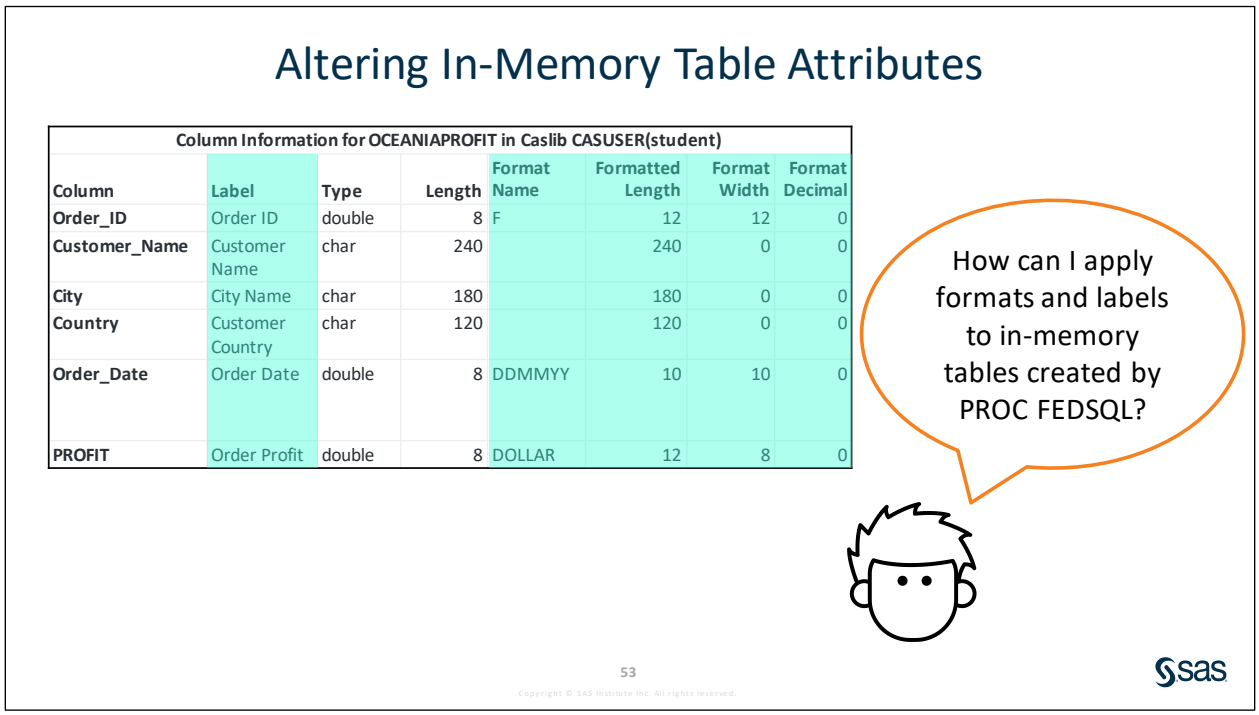
1. Make the following modifications to convert the PROC SQL query to a PROC FEDSQL query that will run in CAS.
   - Change SQL to FEDSQL.
   - Add the SESSREF= option to name the CAS session.
   - Change the input data source to an in-memory table.
   - Replace mnemonic comparisons to operators.
2. What syntax can you use in the WHERE clause to replace the following expression?

```
city ne ' '
```

49



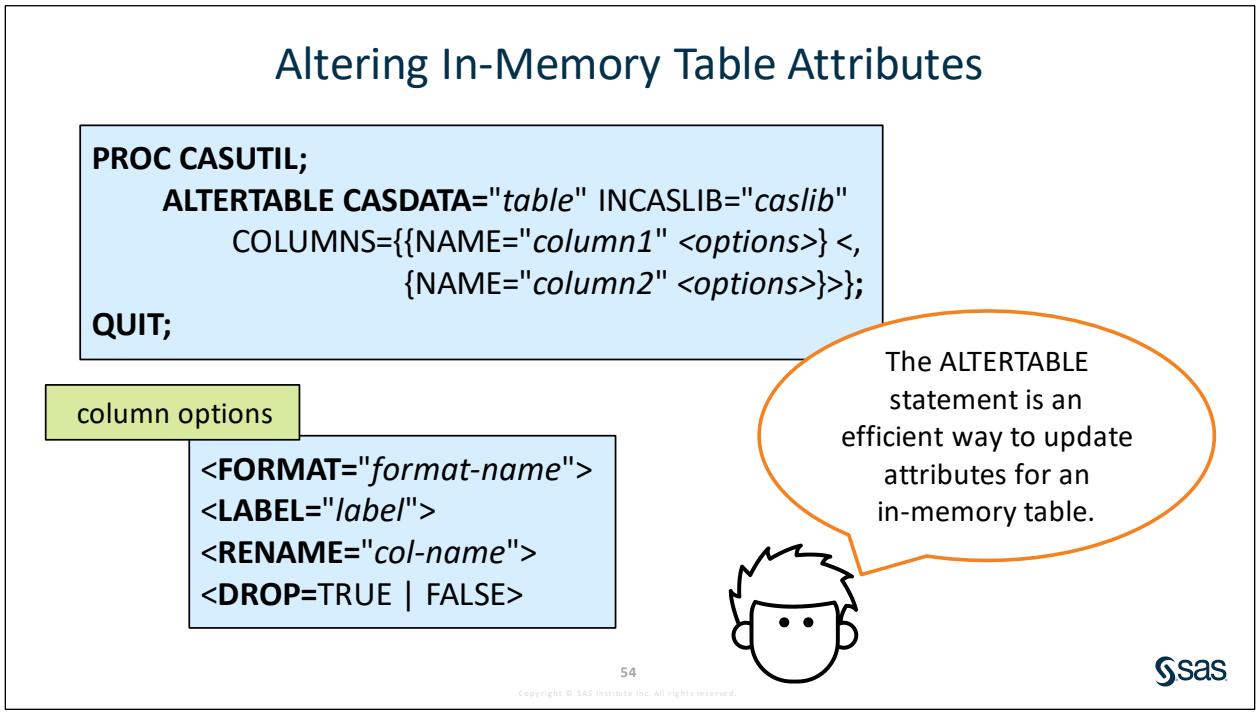
Not all SELECT statement features are supported in PROC FEDSQL. You cannot perform SET operations, use correlated subqueries, use dictionary tables, or create views. The FORMAT= and LABEL= options are SAS enhancements that are also not valid in PROC FEDSQL. This means that you might not be able to write a FedSQL query for every SQL query, and not all queries will be able to run multi-threaded.



Column Information for OCEANIAPROFIT in Caslib CASUSER(student)

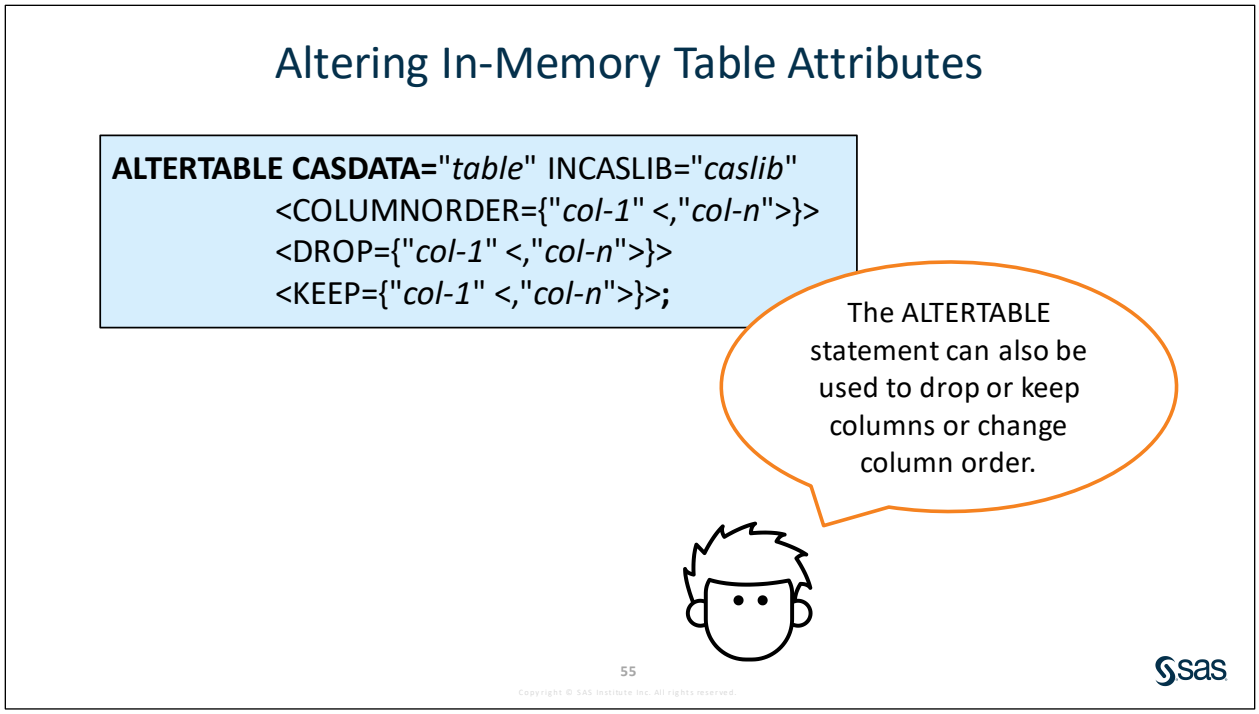
Column	Label	Type	Length	Format Name	Formatted Length	Format Width	Format Decimal
Order_ID	Order ID	double	8	F	12	12	0
Customer_Name	Customer Name	char	240		240	0	0
City	City Name	char	180		180	0	0
Country	Customer Country	char	120		120	0	0
Order_Date	Order Date	double	8	DDMMYY	10	10	0
PROFIT	Order Profit	double	8	DOLLAR	12	8	0

PROC FEDSQL does not support assigning formats and labels directly in a query. However, there are other ways this can be accomplished in in-memory tables.



Starting in SAS Viya 3.5, PROC CASUTIL includes the ALTERTABLE statement. This syntax provides an efficient way to update tables and column metadata without having to read or write the data. The CASDATA= option names the table, and the INCASLIB= option specifies the caslib name. The COLUMNS= option enables you to apply formats and labels, rename columns, and drop columns.

Note that there are at least two sets of required brackets after COLUMNS=. The outer set of brackets surrounds all columns to be modified. Each set of brackets on the inside must include the NAME= option to select a particular column and any options to apply to that column. If multiple columns are modified, a comma separates each set of inner brackets.



The ALTERTABLE statement can also be used to modify table attributes. You can easily change the column order or drop or keep columns.

Creating an In-Memory Table Using PROC FEDSQL

Scenario

Use PROC FEDSQL to create a new in-memory table that includes a calculated column. Modify the attributes of the in-memory table to apply formats and labels.

Program

- **pv03d06.sas**

Syntax

```
PROC FEDSQL SESSREF=cas-session-name;
CREATE TABLE caslib.table-name{OPTIONS REPLACE=TRUE}
SELECT col-name, col-name
      FROM input-table
      WHERE expression
      GROUP BY item-name
      HAVING col-name
      ORDER BY item-name;
QUIT;

PROC CASUTIL;
ALTERTABLE CASDATA="table-name" INCASLIB="caslib"
      COLUMNS={{NAME="column1" <options>} <,
                  {NAME="column2" <options>}>};
QUIT;

Column options include:
   DROP=TRUE | FALSE
   FORMAT="format-name"
   LABEL="label"
   RENAME="table-name"
```

Notes

- The CALCULATED keyword and the FORMAT= option are SAS enhancements to the SQL procedure. They are not compliant with ANSI-1999 standard and cannot be used in the FEDSQL procedure.
- The expression of a calculated column must be repeated in PROC FEDSQL.
- The FEDSQL CREATE TABLE statement table option REPLACE=TRUE is used to overwrite an existing in-memory table.
- Formats and labels can be applied to in-memory tables using the ALTERTABLE statement in PROC CASUTIL.
- PROC FEDSQL does not include syntax to promote an in-memory table after it has been created. Use a PROC CASUTIL step with the PROMOTE statement to promote an existing table.

Demo

Open **pv03d06.sas** from the **demos** folder and perform the following tasks:

Note: If you have not set up the autoexec file in SAS Studio, open and submit **startup.sas** first. If **orders** has not been loaded into memory, uncomment and run the CASUTIL step.

1. The first SQL query produces a Base SAS table named **OceaniaProfit** that calculates **DaystoDelivery**. The second SQL query generates a report based on the **OceaniaProfit** table. Submit the program and examine the results.
2. Make the following modifications to the SQL query to use FedSQL in CAS:
   a. Change **sql** to **fedsql** in the PROC statement and add **sessref=Mysession**.
   b. Change the libref for **OceaniaProfit** and **orders** to **casuser**.
   c. Copy the expression **(RetailPrice-(Cost*Quantity))** in the SELECT list and replace **calculated Profit** in the WHERE clause with the expression.
   d. Cut the ORDER BY clause from the first query. Paste it in the second query after the FROM clause so that the report will be ordered by descending **Profit**.

```
proc fedsql sessref=mysession;
create table casuser.OceaniaProfit as
select Order_ID, Customer_Name, City, Country, Order_Date,
       (RetailPrice-(Cost*Quantity)) as Profit
    from casuser.orders
    where (RetailPrice-(Cost*Quantity))>500
        and Continent = 'Oceania';
select *
    from casuser.OceaniaProfit
    order by Profit desc;
quit;
```

3. Submit the program and examine the results. Notice that **PROFIT** is displayed in capital letters and the data values are not formatted.

   Partial Results

Order ID	Customer Name	City Name	Customer Country	Date Order was placed by Customer	PROFIT
1234249792	Basten, Ms. Chriss	Ascot	Australia	04DEC2013	1414.400000
1230378401	Preece, Ms. Kalli	Wheelers Hill	Australia	10MAR2012	1060.800000
1234890927	Bradford, Mr. Andrey	East Kew	Australia	16FEB2014	1060.500000
1237993374	Moeller, Mr. Steven	Vic 8001	Australia	19DEC2014	1023.000000

4. Run the PROC FEDSQL step again and view the log. The step fails because **casuser.OceaniaProfit** already exists. Add the REPLACE=TRUE option after the table and resubmit the step. Verify that **casuser.OceaniaProfit** is replaced.

```
create table casuser.OceaniaProfit{options replace=true} as
...
```

5. Examine the PROC CASUTIL step. The ALTERTABLE statement applies a label and format to the **Order_Date** column in the **casuser.OceaniaProfit** table. The CONTENTS statement creates a report with the table and column attributes. Run the CASUTIL step and examine the report.
6. Modify the COLUMNS= option in the ALTERTABLE statement to apply the DOLLAR8. format to the **Profit** column. Label the column **Order Profit**.

Note: The outer brackets are required syntax after the COLUMNS= option. Options for each individual column are surrounded by an additional set of brackets and separated by a comma.

```
proc casutil;
    altertable casdata="OceaniaProfit" incaslib="casuser"
               columns={{name="Order_Date" format="ddmmyy10."
               label="Order Date"},
               {name="Profit" format="dollar8."
                label="Order Profit"}};
    contents casdata="OceaniaProfit" incaslib="casuser";
quit;
```

7. Submit the PROC CASUTIL step and view the Column Information report to confirm that the formats and labels are applied to the **Order_Date** and **Profit** columns.

Note: You can also open the **casuser.OceaniaProfit** table from the Libraries section of the navigation pane to view the formatted data values.

Column Information for OCEANIAPROFIT in Caslib CASUSER(student)

Column	Label	Type	Length	Format Name	Formatted Length	Format Width	Format Decimal
Order_ID	Order ID	double	8	F	12	12	0
Customer_Name	Customer Name	char	240		240	0	0
City	City Name	char	180		180	0	0
Country	Customer Country	char	120		120	0	0
Order_Date	Order Date	double	8	DDMMYY	10	10	0
PROFIT	Order Profit	double	8	DOLLAR	12	8	0

	Order_ID	Customer_Name	City	Country	Order_Date	PROFIT
1	1230378401	Preece, Ms. Kalli	Wheelers Hill	Australia	10/03/2012	$1,061
2	1230538356	Sit, Ms. Alina	Townsville Mc	Australia	08/04/2012	$505
3	1230817614	Grebennekov, Ms. Amany	Drummoyne	Australia	29/05/2012	$707
4	1230899723	Tamplin, Mr. Jonathan	Bayswater North	Australia	13/06/2012	$707

End of Demonstration

Practice

If your CAS session has timed out or has been terminated, or SAS Studio timed out, open and submit **startup.sas** before starting the practices.

Level 1

4. Modifying a Simple SQL Query to Run in CAS Using FedSQL

a. Open **pv03p04.sas** from the **practices** folder and submit the program.

```
proc sql;
select Customer_Name,
       Quantity,
       Customer_BirthDate
    from pvbase.orders
    where customer_group contains 'Gold'
        and Continent eq 'Africa'
        and quantity gt 3;
quit;
```

b. Modify the program to run in CAS using FedSQL. Make the following changes:

1) Change **sql** to **fedsql** in the PROC statement.
2) Add the SESSREF= option and specify **mysession** as the value in the PROC statement.
3) Change **pvbase.orders** to **casuser.orders** in the FROM clause.
4) Make the following changes to the WHERE clause:
   a) Use the LIKE operator instead of the CONTAINS operator.

```
where customer_group like '%Gold%'
```

   b) Use the equal operator and the greater than operator instead of the mnemonics.

c. Submit the modified program and examine the results. What is the name of the last customer listed?

Partial Results

Customer Name	Quantity Ordered	Customer Birth Date
Wilkinson, Ms. Rai	4	22MAR1988
Hewitt, Mr. Anton	4	01AUG1968
Eisenberg, Ms. Marietjie	4	08JUL1978
Soicher, Mr. Cordell	4	01FEB1990
Johnstone, Mr. Hans	4	02AUG1958
Poel, Mr. Fanie	5	01MAY1963

Level 2

5. Modifying an SQL Query to Run and Create a Table in CAS Using FedSQL

a. Open **pv03p05.sas** from the **practices** folder and submit the program.

b. Modify the SQL query to use FedSQL to generate **customerTot** as an in-memory table. Remove any syntax that is not permitted in FedSQL. Submit the modified program and examine the output data.

c. Add a PROC CASUTIL step with an ALTERTABLE statement to apply the COMMA8. format to the **TotalCustomers** column and the NLDATE. format to the **CurrentDate** column. Include a CONTENTS statement to view the table and column attributes.

d. Run the program and confirm that the formats and labels are applied in the **casuser.customerTot** table. View the Column Information report. What is the column type for **TotalCustomers**?

City Name	Total Customers	Current Date
Amsterdam	7,603	June 18, 2020
Glasgow	2,566	June 18, 2020
Milano	17,711	June 18, 2020
Firenze	2,059	June 18, 2020
Philadelphia	2,964	June 18, 2020

e. CAS and FedSQL support multiple data types. Search SAS documentation to answer the following question: What is the maximum number of digits that can be stored in each value of **TotalCustomers** given the assigned data type?

Challenge

6. Modifying an SQL Join Query to Join Tables in CAS Using FedSQL

a. Open **pv03p06.sas** from the **practices** folder and submit the program.

b. Modify the SQL query to perform the join in CAS using FedSQL. Create an in-memory table in **Casuser** named **orderdetail**.

Hint: You must load a copy of **pvbase.products** into CAS in order to process the join in CAS using FedSQL.

c. Use PROC CASUTIL to modify the properties of the **Profit** column. Apply the DOLLAR12.2 format and assign **Order Profit** as the label. View the table information to ensure that the column attributes are correctly applied. What is the type of the new **Profit** column?

d. Save the **orderdetail** in-memory table as a SASHDAT table in **Casuser** so that it can be quickly loaded to memory when required. List the data source files in **Casuser** and confirm that **orderdetail.sashdat** is created.

End of Practices

3.3 Solutions

Solutions to Practices

1. Modifying a DATA Step to Run in CAS (Creating New Variables)

```
 /* Part a. */

data work.CouponMailer;
    set pvbase.orders end=eof;
    if quantity in (1,2) then Discount=.15;
    else if quantity in (3,4) then discount=.20;
    else discount=.30;
    FullName=catx(' ', scan(Customer_Name,2,',')
                     , scan(Customer_Name,1));
    MailDate=mdy(12,day(Order_Date),year(today()));
    keep FullName City Postal_Code State_Province Discount
         MailDate;
    if eof then put _threadid_=   _N_=;
    format MailDate worddate.;
run;


 /* Parts b.-c. */

data casuser.CouponMailer;
    set casuser.orders end=eof;
    if quantity in (1,2) then Discount=.15;
    else if quantity in (3,4) then discount=.20;
    else discount=.30;
    FullName=catx(' ', scan(Customer_Name,2,',')
                     , scan(Customer_Name,1));
    MailDate=mdy(12,day(Order_Date),year(today()));
    keep FullName City Postal_Code State_Province Discount
         MailDate;
    if eof then put _threadid_=   _N_=;
    format MailDate nldate.;
run;
```

Note: Actual run times in Base SAS and SAS Viya can vary on the image. However, you should see that SAS Viya processed the rows faster than Base SAS in this practice.

2. Modifying a DATA Step to Run in CAS (Accumulating Totals)

```
data casuser.CustomerCounts;
    set casuser.orders end=eof;
    if scan(customer_Group,3) ='Gold' then GoldCount+1;
    else if scan(customer_Group,1,' ') ='Internet/Catalog'
        then ICCount+1;
    else OtherCount+1;
    if eof then output;
    keep GoldCount ICCount OtherCount;
run;

data casuser.CustomerCounts_Total / single=yes;
    set casuser.CustomerCounts end=eof;
    TotalGold+GoldCount;
    TotalIC+ICCount;
    TotalOther+OtherCount;
    keep Total:;
    if eof=1 then output;
    label TotalGold='Total # Gold Memberships'
          TotalIC='Total # Internet/Catalog Memberships'
          TotalOther='Total # Other Memberships';
    format Total: comma7.;
run;

proc print data=casuser.CustomerCounts_Total label;
    title 'Orion Star Club Membership Counts';
run;
```

3. Modifying a DATA Step to Run in CAS (BY-Group Processing)

```
 /* Part a. */
 /* DATA step processed in Compute Server */

proc sort data=pvbase.orders out=orders;
    by Continent City;
run;

data work.CityTotals(where=(city ne ' '));
    set orders;
    by Continent City;
    if first.City then do;
        TotalCost=0;
        TotalOrders=0;
    end;
```

```
    TotalCost+Cost;
    TotalOrders+1;
    if last.City then output;
    keep Continent City TotalCost TotalOrders;
run;

 /* Parts b.-c. */
 /* DATA step processed in CAS */
data casuser.CityTotals;
    set casuser.orders(where=(city ne ' ')) end=last;
    by Continent City;
    if first.City then do;
        TotalCost=0;
        TotalOrders=0;
    end;
    TotalCost+Cost;
    TotalOrders+1;
    if last.City;
    keep Continent City TotalCost TotalOrders;
    if last then put _threadid_=_nthreads_=;
run;

 /* Parts d.-e. */
options msglevel=i;
title "work.cityTotals";
proc print data=work.CityTotals;
    where city='Abbeville' and continent='North America';
run;
title "casuser.cityTotals";
proc print data=casuser.CityTotals;
    where city='Abbeville' and continent='North America';
run;
options msglevel=n;
```

b. Submit the program and verify that the same number of rows (10,779) were created.

1) Are the results from the Base SAS DATA step and the results from the DATA step that ran in CAS identical? Explain. **No. The order of the rows is different in the DATA step results that processed in CAS. The Base SAS DATA step returned the rows in order by Continent and City. The results in CAS were returned in order based on the thread that processed the DATA step the fastest. The order of output rows is not guaranteed when processing BY groups in CAS.**

2) Can you tell by the results how many threads were used to process the data in CAS? **No**

c. Modify the DATA step to use _THREADID_= and _ NTHREADS _= to add a message in the log showing how many threads were used to process the DATA step in CAS and how many total threads are available on the CAS server.

1) How many threads were used to process the DATA step that ran in CAS? **Five**

2) Were all available CAS threads used for this process? **No**

d. Submit the following code to verify that the calculations for **work.CityTotals** and **casuser.CityTotals** have the same values for *Abbeville, North America*.

e. Examine the results and the log.

1) Can you determine from the log whether CAS processed any portion of the second PROC PRINT step? **Because the MSGLEVEL=I system option is used, the log indicates that the WHERE clause was processed in CAS.**

2) Did both processes produce the same values for **TotalCost** and **TotalOrders** for *Abbeville, North America*? **Yes**

4. Modifying a Simple SQL Query to Run in CAS Using FedSQL

```
proc fedsql sessref=Mysession;
select Customer_Name,
       Quantity,
       Customer_BirthDate
    from casuser.orders
    where customer_group like '%Gold%'
        and Continent = 'Africa'
        and quantity > 3;
quit;
```

What is the name of the last customer listed? **Trollope, Mr. Kibata**

5. Modifying an SQL Query to Run and Create a Table in CAS Using FedSQL

```
 /* Part a. */
proc sql;
create table work.customerTot as
select City,
       count(City) as TotalCustomers "Total Customers",
       today() format=worddate. as CurrentDate "Current Date"
    from pvbase.orders
    group by City
    having TotalCustomers>2000
    order by TotalCustomers descending;
select *
    from work.customerTot;
quit;

 /* Part b. */
proc fedsql sessref=Mysession;
create table casuser.customerTot{options replace=true} as
select city,
       count(city) as TotalCustomers,
       today() as CurrentDate
    from casuser.orders
    group by city
    having count(city)>2000;
quit;


 /* Parts c.-d. */
proc casutil;
    altertable casdata="customerTot" incaslib="casuser"
               columns={{name="TotalCustomers" format="comma8."
                label="Total Customers"},
               {name="CurrentDate" format="nldate."
                label="Current Date"}};
    contents casdata="customerTot" incaslib="casuser";
quit;
```

What is the column type for **TotalCustomers**? **int64**

What is the maximum number of digits that can be stored in each value of **TotalCustomers** given the assigned data type? **19 digits**

6. Modifying an SQL Join Query to Join tables in CAS Using FedSQL

```
 /* Part a. */
proc sql;
create table orderDetail as
select Order_ID, ord.Product_ID, Product_Name, Product_Category,
       (Retailprice-(Cost*Quantity)) as Profit format=dollar12.2
    from pvbase.orders ord left join pvbase.products pc
    on pc.product_id=ord.product_id
    where calculated Profit > 0;
quit;

 /* Part b. */
proc casutil;
    load data=pvbase.products outcaslib="casuser"
         casout="products" replace;
quit;
proc fedsql sessref=mysession;
create table casuser.orderdetail{options replace=true} as
select ord.*, Product_Name, Product_Category, Product_Line,
       Supplier_Name, Supplier_Country,
       (ord.Retailprice-(Cost*Quantity)) as Profit
    from casuser.orders ord left join casuser.products pc
    on pc.product_id=ord.product_id
    where (Retailprice-(Cost*Quantity)) > 0;
quit;

 /* Part c. */
proc casutil;
    altertable casdata="orderdetail" incaslib="casuser"
               columns={{name="Profit" format="dollar12.2"
                         label="Order Profit"}};
    contents casdata="orderdetail" incaslib="casuser";
quit;

 /* Part d. */
proc casutil;
    save casdata="orderdetail" casout="orderdetail" replace;
    list files;
quit;
```

What is the type of the new **Profit** column? **Double**

End of Solutions

Solutions to Activities and Questions

3.01 Activity – Correct Answer

3. How many threads are available? **It depends on your SAS Viya environment.**

```
data _null_ / sessref="MySession";
```

```
NOTE: Running DATA step in Cloud Analytic Services.
Processed on _THREADID_=8 _NTHREADS_=16
Processed on _THREADID_=3 _NTHREADS_=16
Processed on _THREADID_=7 _NTHREADS_=16
...
```

5. Where did the DATA step execute? How many threads were used to process the DATA step?

   The DATA step ran in CAS, but only on a single thread.

```
data _null_ / sessref="MySession" single=yes;
```

```
NOTE: Running DATA step in Cloud Analytic Services.
Processed on _THREADID_=1 _NTHREADS_=1
```

12

sas

3.02 Activity – Correct Answer

Which DATA step had the shortest real time for processing?

Multi-threaded was faster, but it produced incorrect results.

```
NOTE: Running DATA step in Cloud Analytic Services.
real time 0.10 seconds
cpu time 0.00 seconds
```

multi-threaded

```
NOTE: Running DATA step in Cloud Analytic Services.
real time 0.55 seconds
cpu time 0.01 seconds
```

single-threaded

Note: Your time values might differ.

22

sas

3.03 Question – Correct Answer

Were all of the available threads used in CAS to process this DATA step?

No. There are 16 threads available on the CAS server. CAS distributed the data among 4 of the threads for processing.

```
NOTE: Running DATA step in Cloud Analytic Services.
NOTE: The DATA step will run in multiple threads.
_NTHREADS_=16 _THREADID_=11 _N_=770
_NTHREADS_=16 _THREADID_=15 _N_=1110
_NTHREADS_=16 _THREADID_=12 _N_=235708
_NTHREADS_=16 _THREADID_=13 _N_=714081
```

Note: The number of threads used and available may differ depending on your environment.

30

continued...

3.04 Activity – Correct Answer

Which portion of the DATA step was processed in CAS?

The WHERE statement

```
86 where ordertype=3;
NOTE: The where clause is being processed by Cloud Analytic Services.
```

Why did the entire DATA step not process in CAS? **The output table in the DATA statement did not reference an in-memory table.**

```
NOTE: 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.
NOTE: Could not execute DATA step code in Cloud Analytic Services.
      Running DATA step in the SAS client.
```

40

3.04 Activity – Correct Answer

4. Modify the DATA step to run in CAS and resubmit the step. Confirm in the log that the entire step processes in CAS.

change the output table to an in-memory table

```
82 options msglevel=i;
83
84 data casuser.retail;
85 set casuser.orders;
86 where ordertype=3;
NOTE: The where clause is being processed by Cloud Analytic Services.
87 Profit=RetailPrice-(Cost*Quantity);
88 run;
NOTE: Running DATA step in Cloud Analytic Services.
NOTE: The DATA step will run in multiple threads.
```

41

continued...

3.05 Activity – Correct Answer

1. Make the following modifications to convert the PROC SQL query to a PROC FEDSQL query that will run in CAS.

```
options msglevel=i
proc fedsql sessref=mysession;
select distinct Customer_Name, City
   from casuser.orders
   where Continent = 'Africa'
         and city <> ' '
         and cost >= 500
   order by City;
quit;
options msglevel=n;
```

Customer Name	City Name
Browne, Mr. Wilhelm	Johannesburg
Baga, Ms. Aysha	Linden
Morton, Ms. Diana	Pretoria
Wigram, Mr. Mark	Saldanha Bay
Ouedrago, Ms. Amanda	Wirral

50

3.05 Activity – Correct Answer

2. What syntax can you use in the WHERE clause to replace the following expression?

```
city ne ' '
```

FEDSQL alternatives:

```
city <> ' '
```

```
city <> null
```

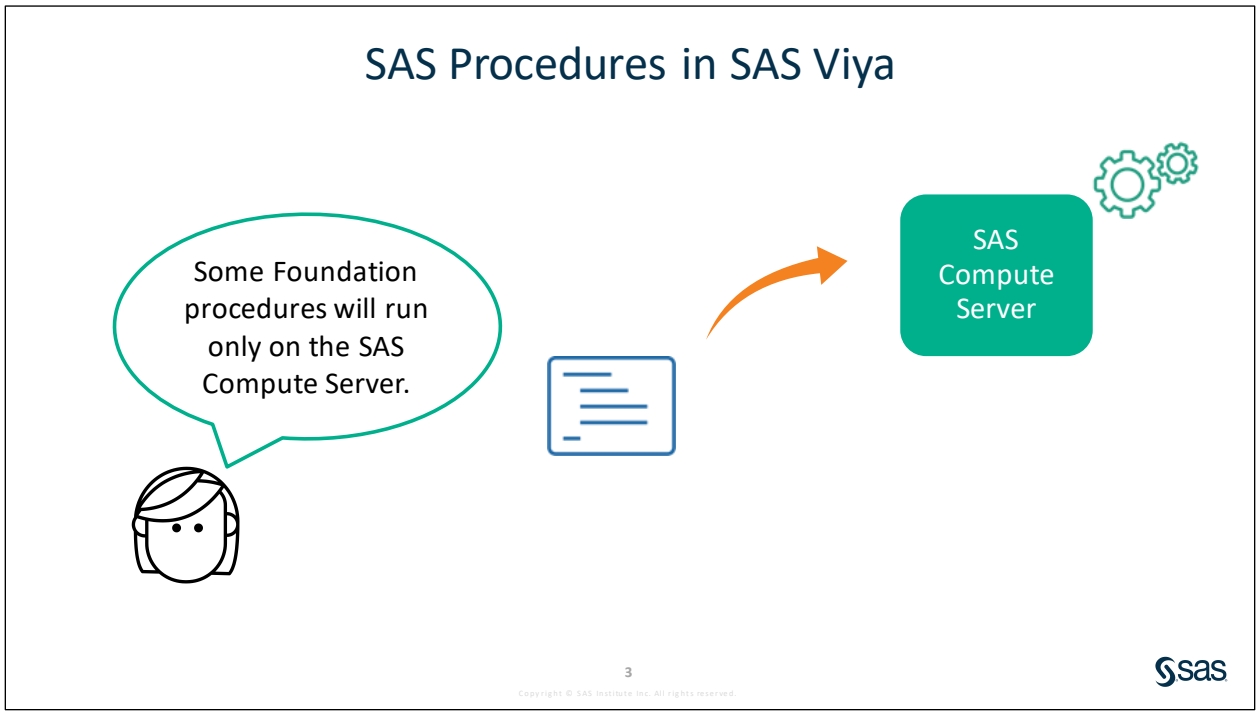
```
city is not null
```

```
city is not missing
```

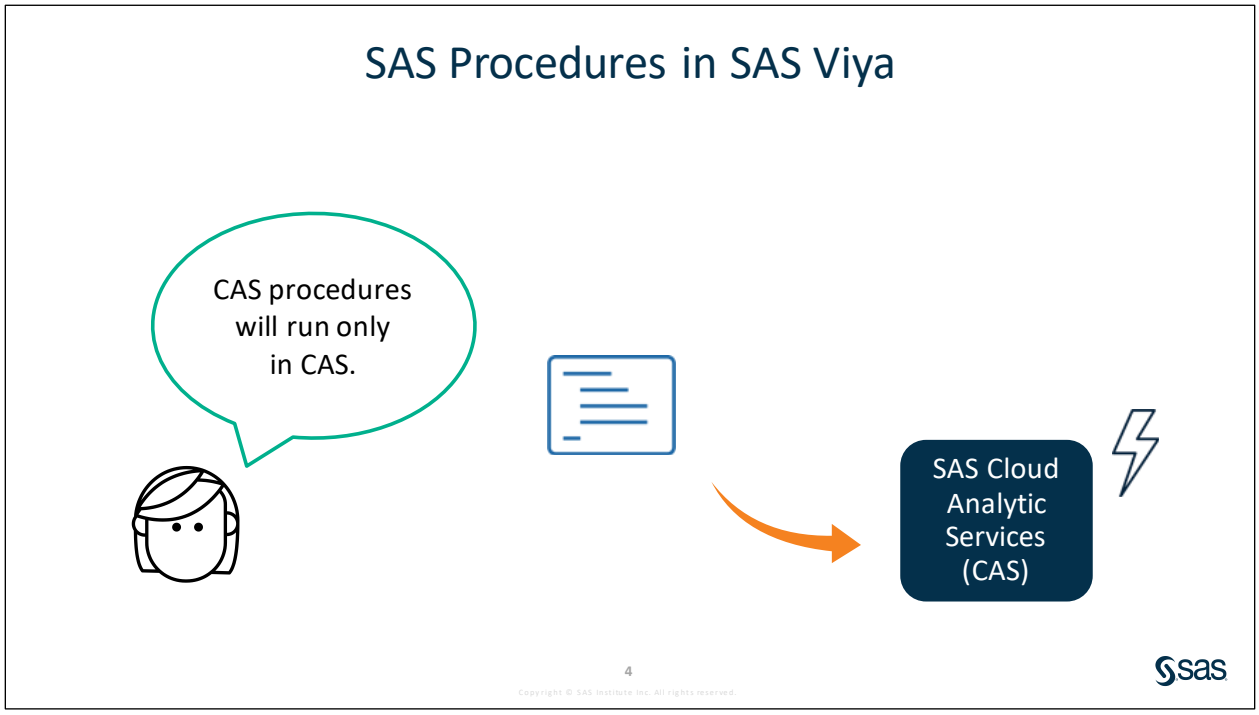
51

Lesson 4 Using CAS-Enabled Procedures and Formats

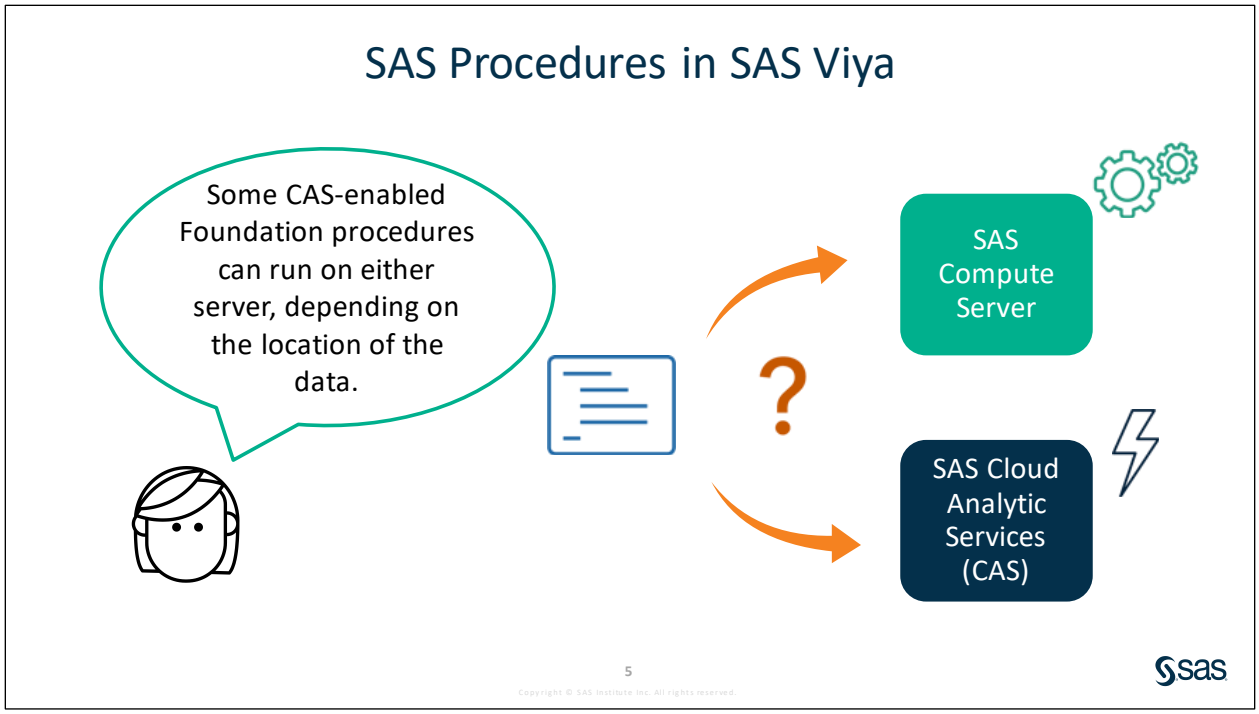
4.1 CAS-Enabled Procedures



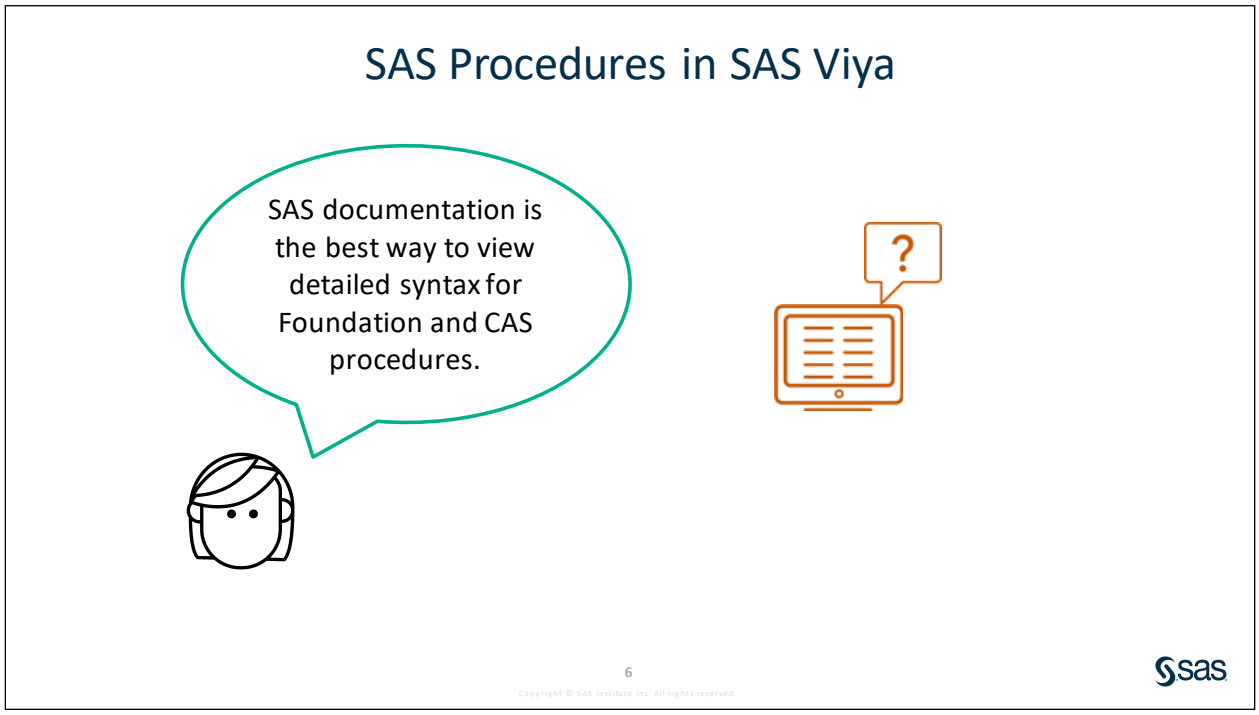
There are some Foundation procedures that will run only on the SAS Compute Server.



There is a collection of new procedures that run only in CAS on in-memory tables.



There is another group of procedures that can run on either server, depending on where the data in located and the statements and options used.



With so many procedures available in SAS, the best place to go to learn about syntax and considerations for procedures is SAS documentation.

SAS Viya Procedures Documentation

1. In a browser window, visit the SAS Help Center at http://documentation.sas.com/. Click **Programming SAS 9.4 and SAS Viya 3.5**.
2. In the **SAS Viya** section, click **Introduction to SAS Viya Programming**.
3. In the left navigation pane, select **SAS Cloud Analytic Services Procedures**. These procedures are available in all Viya installations. They use CAS tables and capabilities, ensuring full use of parallel processing. The MDSUMMARY procedure is similar to the MEANS procedure, except it is optimized to run only in CAS.
4. In the left navigation pane, select **SAS Viya Foundation Procedures**. The table of SAS Foundation procedures indicates whether they run in CAS. For example, PROC APPEND cannot process in CAS, but PROC CONTENTS and PROC COPY can.

   Note: This subset of SAS 9.4 Foundation procedures is available if you have the most basic license, which includes SAS Viya and SAS Visual Analytics. If you have any other Viya products licensed, you have access to all SAS 9.4 Foundation procedures.
5. Find the MEANS procedure in the table. PROC MEANS will run in CAS if the DATA= option specifies a data set with a CAS LIBNAME engine libref. To learn more about programming requirements, right-click **MEANS** and select **Open link in new tab**.
6. On the MEANS Procedure page, click **Concepts** ⇨ **CAS Processing for PROC MEANS**. This section provides instructions for ensuring that PROC MEANS code is CAS-enabled. For example, notice that there is a list of statistics that are supported in CAS and a list of statistics that are not currently supported.
7. Return to the browser tab with the **Introduction to SAS Viya Programming** documentation. Additional CAS procedures are available depending on the Viya products licensed and installed. Click **SAS Visual Statistics Procedures**. These CAS procedures are used with the Visual Statistics application but are also available to use in SAS programs. For example, the FREQTAB procedure is similar to the foundation FREQ procedure, but it exclusively runs in CAS on in-memory data.

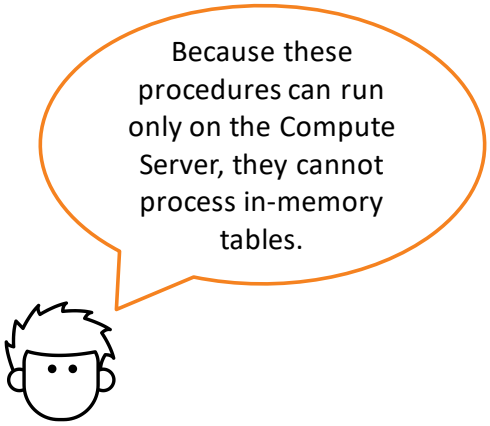
End of Demonstration

4.01 Activity

1. In a browser, go to http://documentation.sas.com/. Click **Programming SAS 9.4 and SAS Viya 3.5**.
2. In the SAS Viya section, click **Introduction to SAS® Viya® 3.5 Programming**.
3. In the left navigation pane, click **SAS Viya Foundation Procedures**.
4. Examine the table that lists foundation procedures and indicates whether they can use CAS processing.
5. Can the MEANS procedure process in CAS?
6. Can the SQL procedure process in CAS?

8

SAS Foundation Procedures



FREQ | SQL

UNIVARIATE | APPEND

SG graphics (SGPLOT, SGPIE, SGSCATTER, and so on)

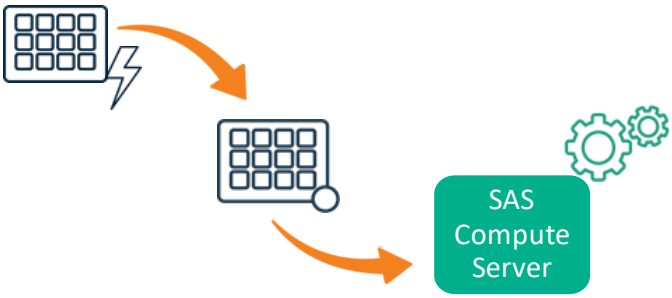
10

The FREQ, SQL, UNIVARIATE, and APPEND procedures and the SG graphics procedures are examples of Foundation procedures that will run on the Compute Server. These steps cannot process in-memory tables directly.

SAS Foundation Procedures

```
proc freq data=casuser.orders;
    tables Continent;
run;
```

If an in-memory table is referenced in a procedure that runs only on the Compute Server, a copy of the data is created for processing.



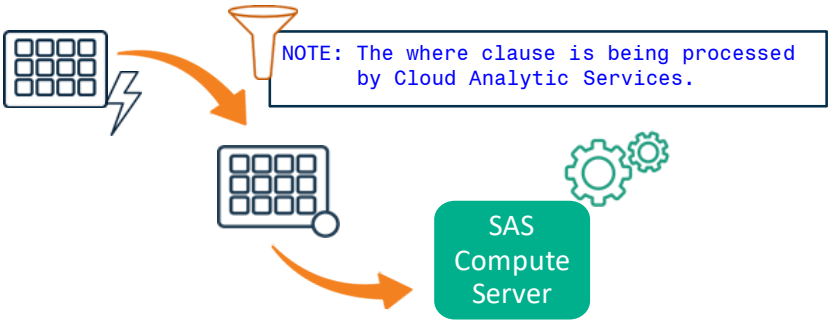
11

If an in-memory table is referenced in a procedure that runs only on the Compute Server, SAS automatically creates a temporary .sas7bdat table for processing on the Compute Server.

SAS Foundation Procedures

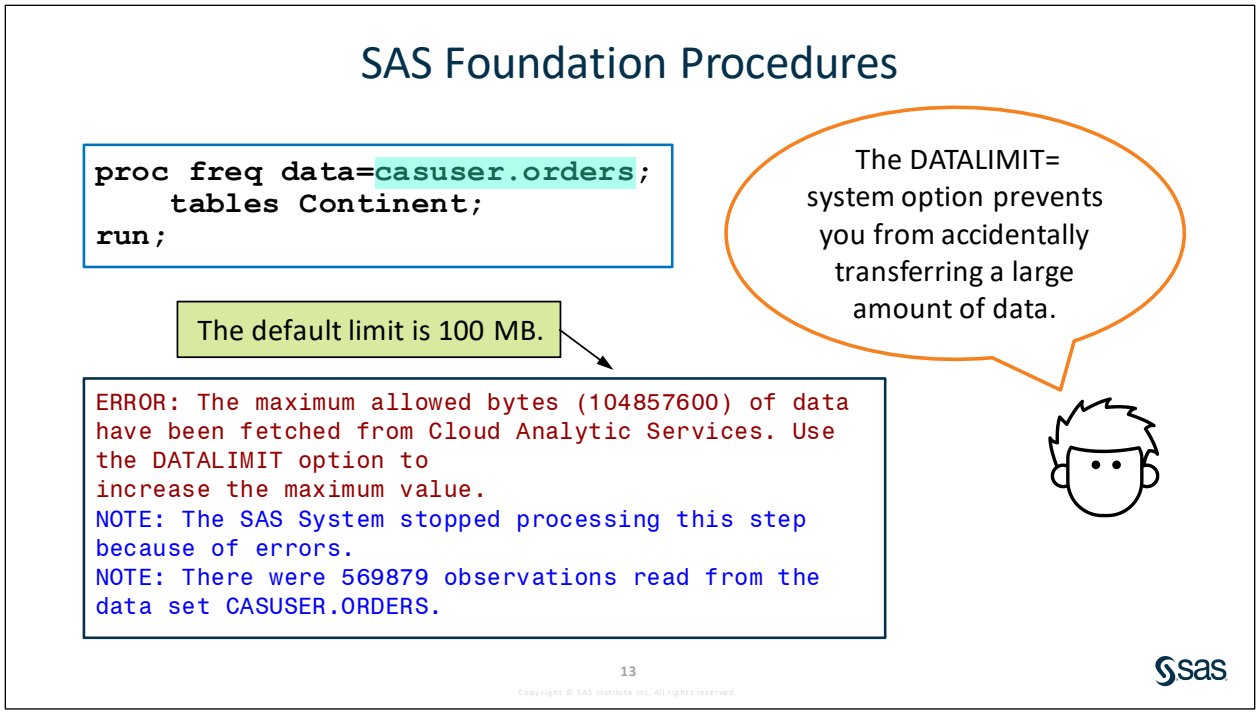
```
proc freq data=casuser.orders;
    tables Continent;
    where OrderType in (2,3);
run;
```

If a WHERE statement is included, CAS filters the in-memory data before sending it to the Compute Server for processing.

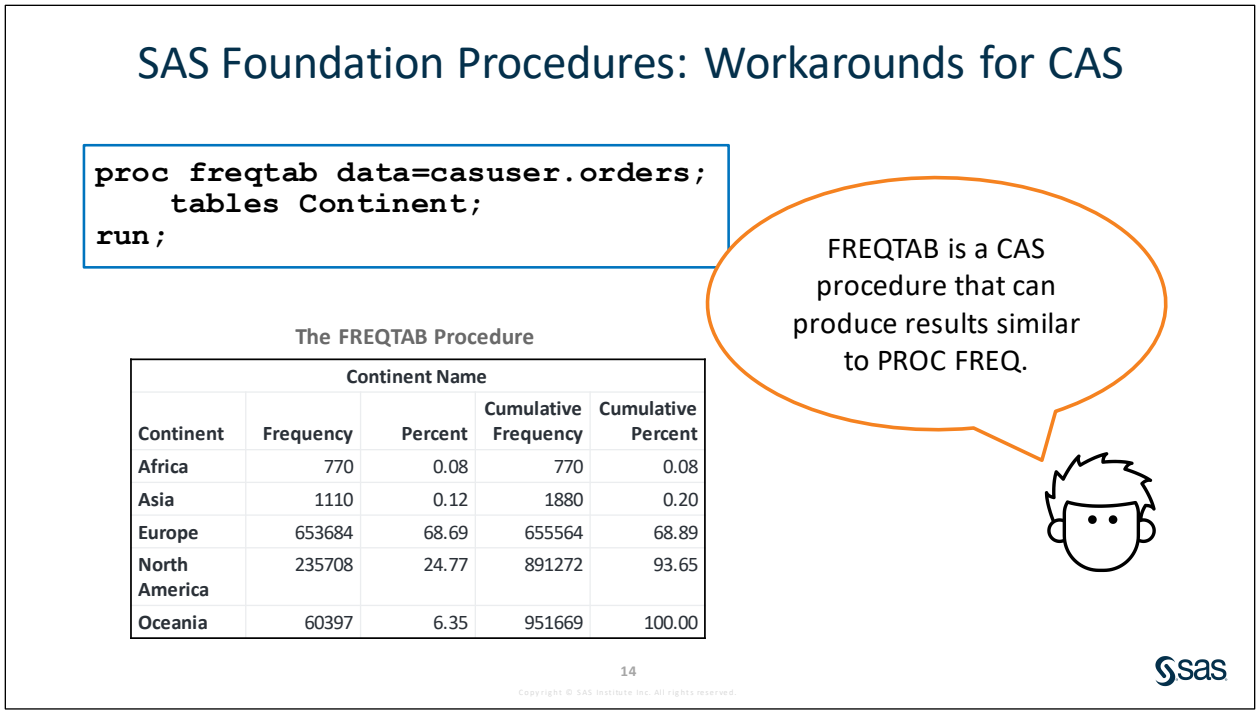


12

If data transfer from CAS to Compute Server is required, SAS attempts to optimize the process and prevent you from moving a large amount of data. If a WHERE statement is included, CAS filters the in-memory data before sending it to the Compute Server.



The DATALIMIT= system option sets a threshold for the maximum allowed bytes of data that can be transferred from CAS. By default, the limit is 100 MB. This prevents you from accidentally moving large in-memory tables to the Compute Server behind the scenes.

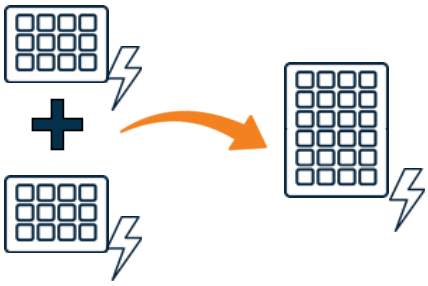


Obviously, if we have in-memory tables, we want that data to be processed in CAS. For many Compute Server procedures, there is an available CAS procedure to accomplish a similar result. For example, instead of using PROC FREQ, you can use PROC FREQTAB on an in-memory table. PROC FREQTAB has similar syntax to PROC FREQ.

Note: The FREQTAB procedure is available if you have SAS Visual Statistics licensed.

SAS Foundation Procedures: Workarounds for CAS

```
data casuser.orders(append=yes);
    set casuser.orders_new;
run;
```



PROC APPEND runs only on the Compute Server, but you can use the DATA step with in-memory tables to produce similar results using CAS.

15

PROC APPEND runs only on the Compute Server, but you can use the DATA step with in-memory tables to produce similar results using CAS. The APPEND=YES option in the DATA statement reads the rows from the table listed in the SET statement and appends them to the table in the DATA statement. In this DATA step, the rows from **casuser.orders_new** are appended to the existing **casuser.orders** in-memory table.

SAS Foundation Procedures: Workarounds for CAS

```
data casuser.orders(append=yes);
    set casuser.orders_new;
run;
```

Rows are read from the table in the SET statement and appended to the table in the DATA statement.

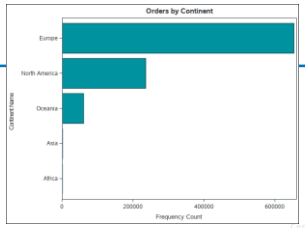
```
NOTE: Running DATA step in Cloud Analytic Services.
NOTE: The DATA step will run in multiple threads.
NOTE: There were 10000 observations read from the table
      ORDERS_NEW in caslib CASUSER(student).
NOTE: The table orders in caslib CASUSER(student) has
      961669 observations and 24 variables.
NOTE: The APPEND operation for table orders in caslib
      CASUSER(student) added 10000 observations.
```

16

The log indicates that the step processed in CAS. With this DATA step, 1,000 rows were read from **casuser.orders_new** and appended to **casuser.orders**.

SAS Foundation Procedures: Workarounds for CAS

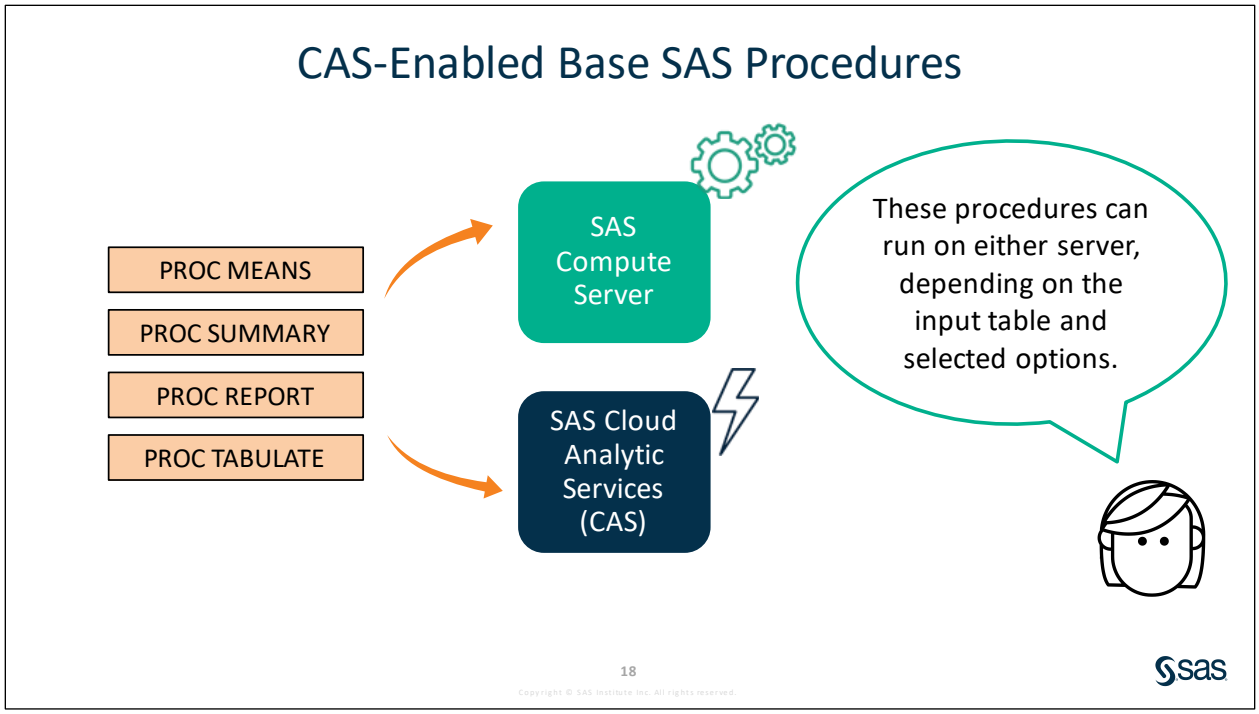
```
proc freqtab data=casuser.orders noprint;
    tables Continent /
           out=casuser.cont_n;
run;

title "Orders by Continent";
proc sgplot data=casuser.cont_n;
    hbar Continent /
         response=Count
         categoryorder=respdesc;
run;
title;
```



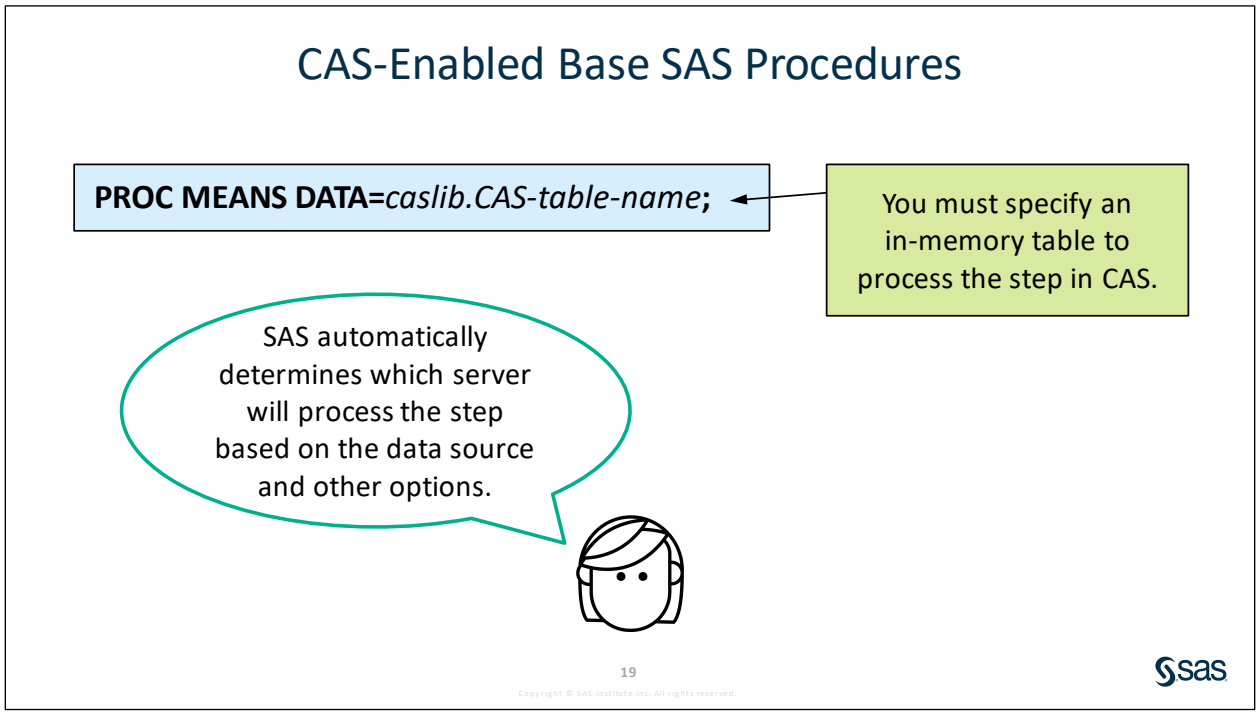
To efficiently produce graphics, first summarize in-memory tables in CAS and then use ODS Graphics procedures on the Compute Server.

17

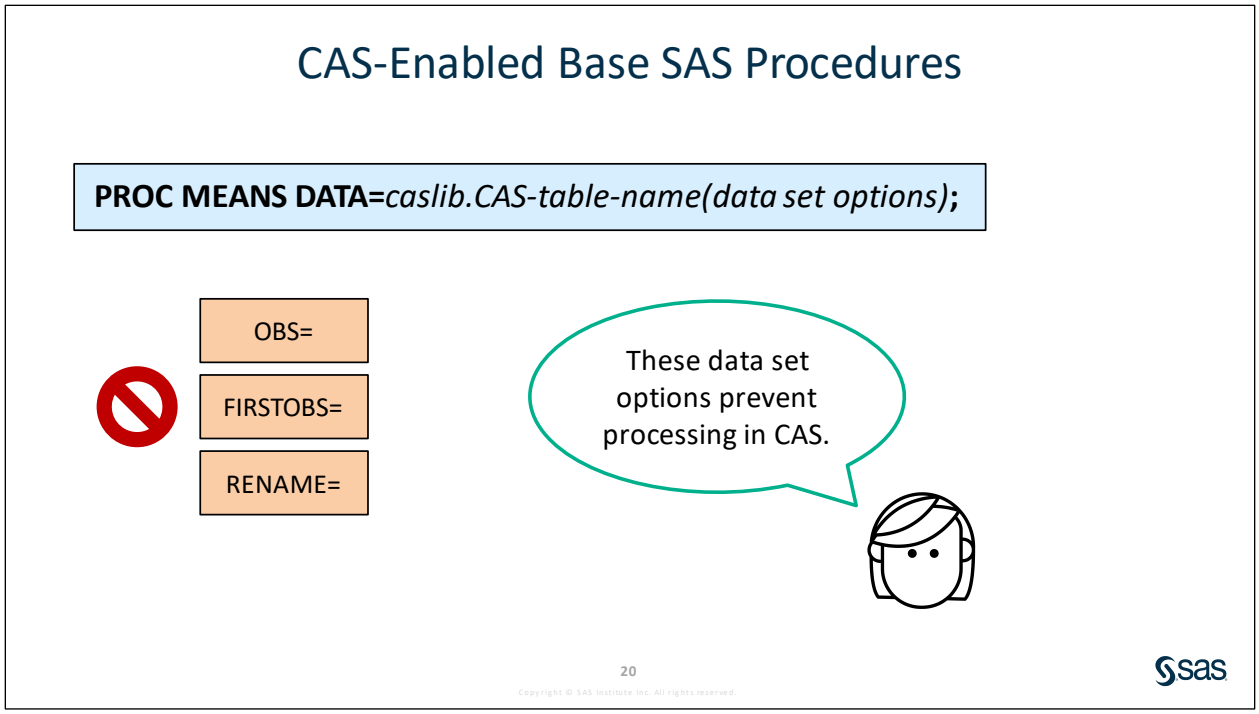
Graphics procedures, including SGPLOT, cannot process in-memory data directly. An efficient workaround is to use CAS to summarize the data first. For example, this PROC FREQTAB step produces an output in-memory table with frequency counts for each value of **Continent**. Then the PROC SGPLOT step produces a bar chart based on the output **casuser.cont_n** table. Remember, a temporary copy of **casuser.cont_n** is created on the Compute Server for processing, but the data transfer will be insignificant because the table is very small.



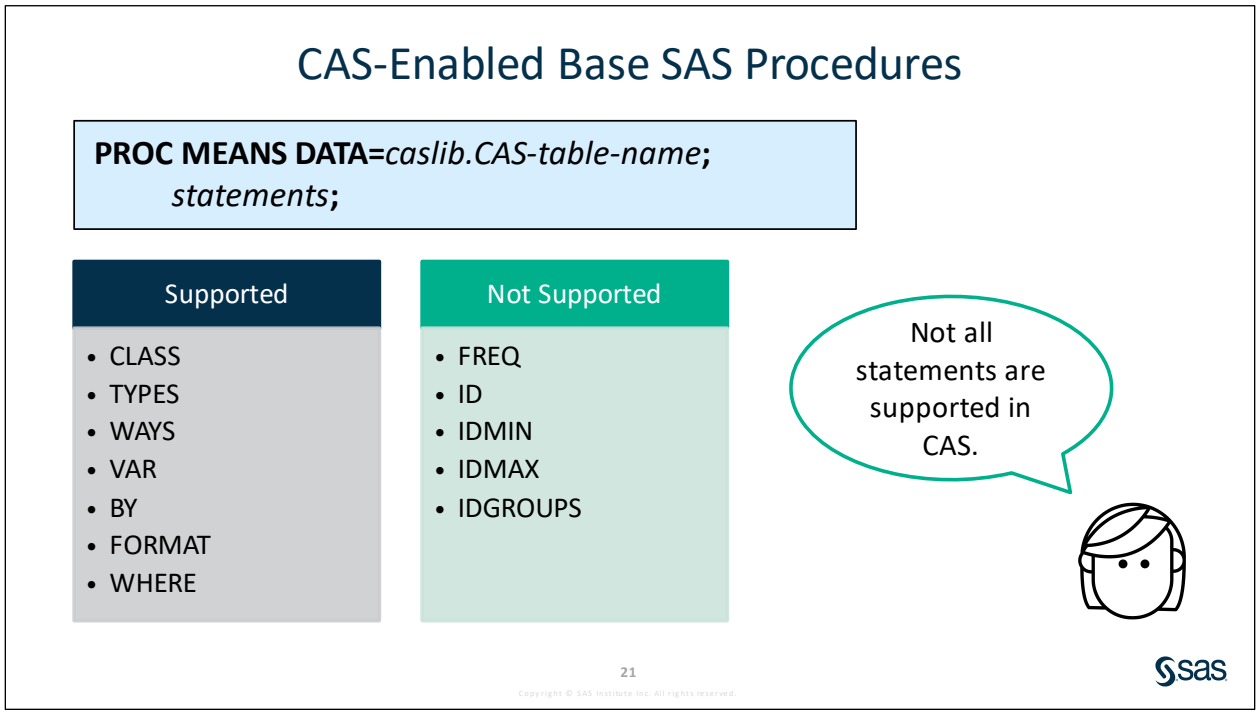
MEANS, SUMMARY, REPORT, and TABULATE are all CAS-enabled procedures. This means that they can run on either server. Which server performs the processing depends on the data and options used in the step.



If the input table is anything other than an in-memory table, then the step processes on the Compute Server. If the input data is an in-memory table and all statements and options are CAS compliant, then SAS automatically converts the step behind the scenes into CAS actions.



You also want to avoid using the OBS=, FIRSTOBS=, and RENAME= data set options. These options prevent processing in CAS.



Not all PROC MEANS and PROC SUMMARY statements are supported in CAS. Use the supported statements in order to process the code in CAS. If you use a statement that is not supported in CAS, SAS processes as much as possible in CAS first, and then transfers the data to the Compute Server to produce the final results.

CAS-Enabled Base SAS Procedures

```
PROC MEANS DATA=caslib.CAS-table-name;
      BY column;
...
```

If you use a BY statement, you don't need to pre-sort the data if the step will process in CAS!

22

But remember that if the BY statement is used in PROC MEANS or any other procedure processing in CAS, the data does not need to be sorted.

CAS-Enabled Base SAS Procedures

```
PROC MEANS DATA=caslib.CAS-table-name statistics;
```

Supported Statistics

- N, NMISS, MIN, MAX, RANGE, SUM, SUMWGT, MEAN, CSS, USS, VAR, STD, STDERR, PRET, UCLM, LCLM, CLM, and CV

Not Supported

- SKEW, KURT, P1, P5, P10, P20, P25/Q1, P30, P40, P50/MEDIAN, P60, P70, P75/Q3, P80, P90, P95, P99, and MODE

23

Not all statistics are supported in CAS for the MEANS procedure. If the procedure uses statistics that are not supported, processing of intermediate aggregates must still be performed by the MEANS procedure on the Compute Server.

If you want to guarantee that all processing is performed in CAS, you can invoke an appropriate CAS action directly, using a CAS-specific PROC or PROC CAS and CASL.

4.02 Activity

Open **pv04a02.sas** from the **activities** folder and perform the following tasks:

1. Notice that the input table is a Base SAS table. Run the program and examine the report.
2. Change the input table to **casuser.orders** to reference an in-memory table. Run the program and examine the log. What is the name of the CAS action used to perform the summarization?

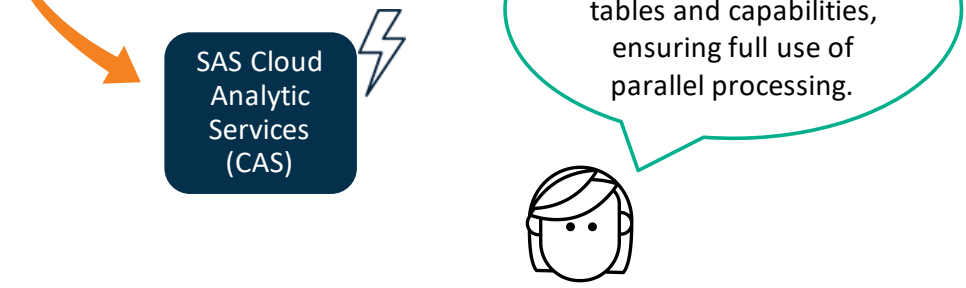
```
proc means data=casuser.orders sum mean;
```

3. Add the MEDIAN option to the PROC MEANS statement and run the program. Does the program run successfully?

24

CAS Procedures

```
PROC MDSUMMARY DATA=caslib.CAS-table-name;
     OUTPUT OUT=caslib.CAS-table-name <options>;
     <statements>
RUN;
```



27

If you want to ensure that all processing is performed in CAS, you can use the MDSUMMARY procedure to compute basic descriptive statistics. The MDSUMMARY procedure uses only CAS tables and does not display output. The results are stored in an output table, and you can display the results of the output table by using the PRINT procedure.

Let's look at using the MDSUMMARY procedure to produce descriptive statistics.

Producing Descriptive Statistics in CAS Using PROC MDSUMMARY

Scenario

Use the MDSUMMARY procedure to create a CAS table with descriptive statistics.

Program

- **pv04d01.sas**

Syntax

```
PROC MDSUMMARY DATA=libref.table-name <NTHREADS=integer>;
     VAR <variable-list>;
     OUTPUT OUT=table-name;
     GROUPBY variable-list </ OUT=table-name>;
RUN;
```

Notes

- The OUTPUT statement is required unless you use the GROUPBY statement and the OUT= option in the GROUPBY statement.

Demo

Open **pv04d01.sas** from the **demos** folder and perform the following tasks:

Note: If you have not set up the autoexec file in SAS Studio, open and submit **startup.sas** first. If **orders** has not been loaded into memory, uncomment and run the CASUTIL step.

1. MDSUMMARY is a CAS procedure that summarizes in-memory data. Submit the program and examine the error message in the log.

```
89   proc mdsummary data=casuser.orders;
90       var RetailPrice;
91   run;
ERROR: The proc requires an OUT= on each GROUPBY statement, or on an OUTPUT statement when no
more than one GROUPBY statement is specified.
```

2. The MDSUMMARY procedure does not produce printed output, so you must include an OUTPUT statement to write the results to an in-memory table. Return to the program and add the OUTPUT statement to create an output table named **casuser.order_all**.

```
proc mdsummary data=casuser.orders;
    var RetailPrice;
    output out=casuser.orders_all;
run;
```

3. Submit the program and examine the statistics on the Output Data tab.

   Partial Output Data

Column	_Min_	_Max_	_NObs_	_NMi...	_Mean_	_Sum_
RetailPrice	0.625	9385.8	951669	0	139.9609407	133196488.48

   Note: The statistics produced by PROC MDSUMMARY cannot be modified.

4. Add a GROUPBY statement to calculate statistics for each unique value of **Country**. Submit the program and examine the Output Data tab. Notice that there are 47 rows in the table.

```
proc mdsummary data=casuser.orders;
    var RetailPrice;
    groupby Country;
    output out=casuser.orders_country;
run;
```

5. Multiple GROUPBY statements can be used with the OUT= option to create multiple output tables. Delete the OUTPUT statement and add the following GROUPBY statements. The first GROUPBY statement will include statistics calculated for the entire input table. The second GROUPBY statement will calculate statistics for each value of **Country**.

```
proc mdsummary data=casuser.orders;
    var RetailPrice;
    groupby / out=casuser.orders_all;
    groupby Country / out=casuser.orders_country;
run;
```

6. PROC PRINT can be used to produce a report with a subset of the statistics and formatted values. Submit the program and examine the **casuser.orders_all** and **casuser.orders_country** tables on the Output Data tab. Uncomment the PROC PRINT step and submit the program.

   Note: PROC PRINT processes on the Compute Server, so the in-memory data must be transferred. No errors are generated in this process because the data is less than the default limit of 100MB.

Customer Country	Number of Orders	Average Retail Price per Order	Total Retail Price
Andorra	73	$192	$14,035
Australia	60320	$131	$7,894,103
Austria	1509	$140	$211,332
Belgium	24943	$126	$3,137,552
Benin	12	$68	$818
Bulgaria	14	$80	$1,117
Canada	3450	$190	$655,740
China	3	$50	$149
Croatia	64	$124	$7,953
Czech Republic	32	$346	$11,072

End of Demonstration

CAS-Enabled Base SAS Procedures

PROC REPORT DATA=*caslib.CAS-table-name***;**

PROC TABULATE DATA=*caslib.CAS-table-name***;**

Like PROC MEANS, these reporting procedures must use an in-memory table and CAS-compliant options to process in CAS.

Supported Statistics

- CSS, RANGE, CV, STDERR, N, MAX, SUM, MEAN, SUMWGT, MIN, STD, USS, NMISS, and VAR
- LCLM and UCLM (TABULATE only)

29

PROC REPORT and PROC TABULATE summarization can be executed on the CAS server if your input data set originates from CAS. Both procedures have a long list of statistics that can be used to analyze data. However, not all statistics are supported in CAS. If you use the supported statistics listed here, the analysis can be performed in CAS.

CAS-Enabled Base SAS Procedures

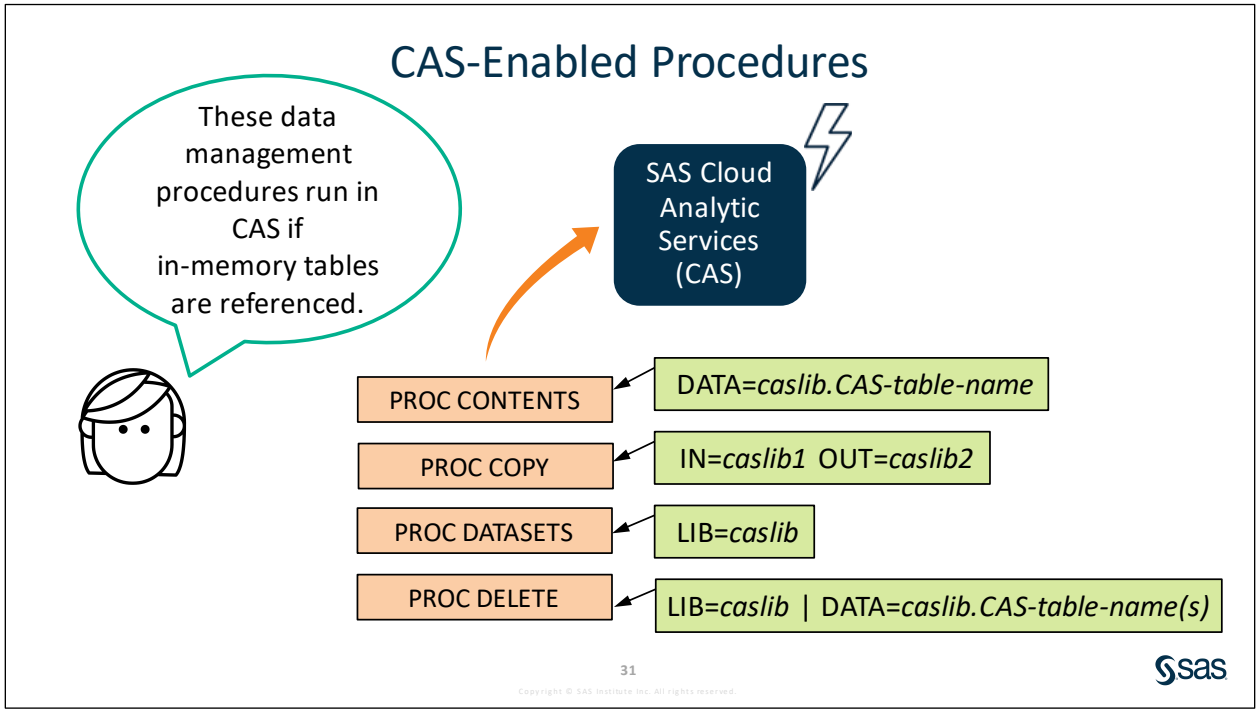
PROC REPORT DATA=*caslib.CAS-table-name;*
DEFINE *column* / **DISPLAY**|**ORDER;**

SAS Compute Server

If a DISPLAY or ORDER variable is used in the report, then PROC REPORT runs on the Compute Server.

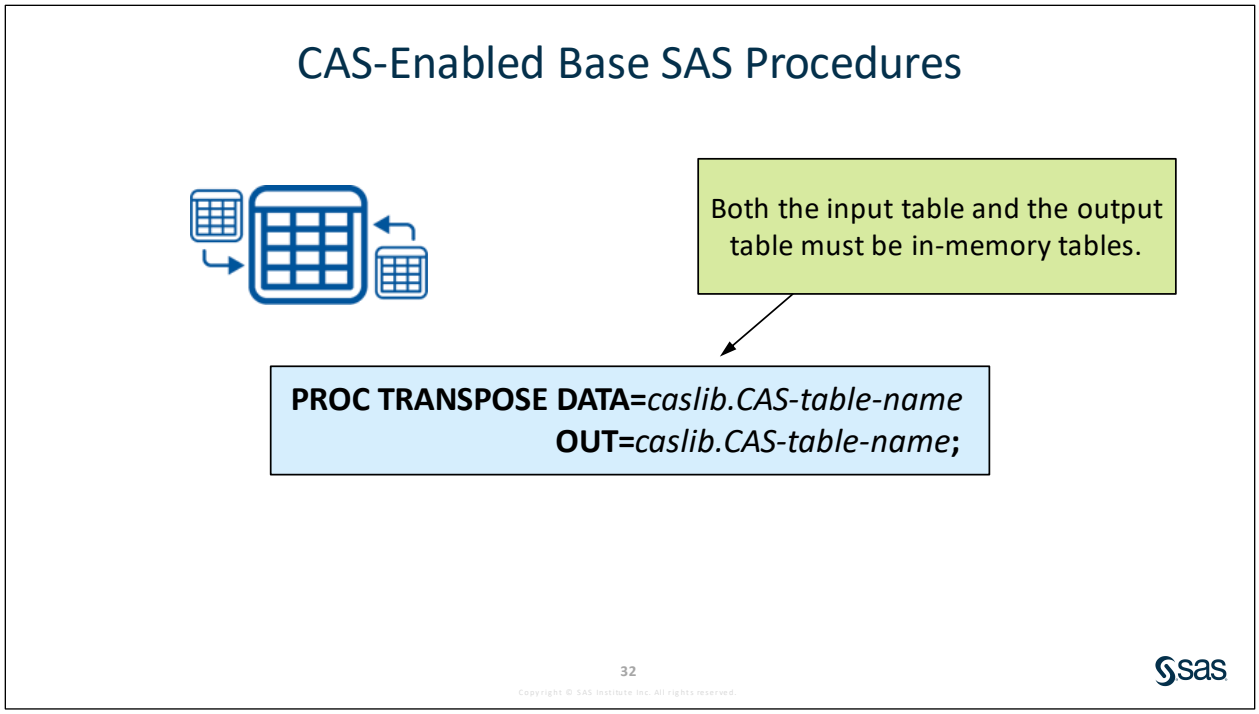
30

If PROC REPORT contains variables with a usage type of DISPLAY or ORDER, the step attempts to run on the Compute Server rather than in CAS.



Procedures that perform data management and show table metadata data, like PROC CONTENTS or PROC DATASETS, and procedures like PROC COPY and PROC DELETE that copy or delete tables can run in CAS. The key to these CAS-enabled procedures is to reference in-memory tables.

- In PROC CONTENTS, you specify the in-memory table name in the DATA= option.
- In PROC COPY, you specify the caslib library reference to copy in-memory tables from one caslib to another caslib.
- In PROC DATASETS, you specify the caslib library reference with the LIB= option.
- In PROC DELETE, you specify the caslib in the LIB= option or the name of the in-memory CAS table that you want to delete in the DATA= option.



Running PROC TRANSPOSE in CAS has several advantages over processing within the Compute Server—namely, reduced network traffic and faster processing. If your input table originates from CAS and your output table is directed back to the CAS server, then the transposition can be performed entirely on the server.

Practice

If you have reset your SAS or CAS session, submit **startup.sas** before starting the practice.

Level 1

1. Using the MDSUMMARY Procedure to Generate Statistics in CAS

a. Open **pv04p01.sas** from the **practices** folder. Submit the PROC CASUTIL step and confirm that the **employees** table is loaded to the **casuser** caslib.

b. Add a VAR statement to calculate summary statistics for **Salary**.

c. Add an OUTPUT statement to produce an in-memory table named **casuser.emp_summary**.

d. Add a GROUPBY statement to calculate statistics for each unique value of **Department**. Run the PROC MDSUMMARY step and confirm that the output table includes 17 rows.

e. Delete the OUTPUT statement. Modify the existing GROUPBY statement to use the OUT= option to write the output table to **casuser.dept_summary**. Add a second GROUPBY statement to calculate statistics for each value of **Country** and create an output table named **casuser.country_summary**.

f. Submit the MDSUMMARY and PRINT steps to generate a report for the statistics grouped by **Country**. What is the mean salary for **Country**=*GB*?

Partial Results

Average Salary by Country

Obs	Country	_Min_	_Max_	_NObs_	_NMiss_	_Mean_
1	AU	$30,125	$220,104	108	0	$40,205
2	BE	$31,306	$195,306	60	0	$40,116
3	DE	$30,038	$189,925	98	0	$39,467
4	DK	$31,456	$204,235	51	0	$41,408
5	ES	$28,381	$220,475	66	0	$40,694
6	FR	$29,481	$241,175	98	0	$41,021

Level 2

2. Transposing an In-Memory Table in CAS

a. Open the **pvbase.qtr_sales** table. Notice that there are four rows for each value of **Product_ID**, one row for each quarter.

b. Open the **pv04p02.sas** program from the **practices** folder. The PROC TRANSPOSE step rotates the table to create separate columns for each quarter. Run the program and examine the output table.

c. Load the **pvbase.qtr_sales** table into the **casuser** caslib. Modify the PROC TRANSPOSE code so that the step processes in CAS.

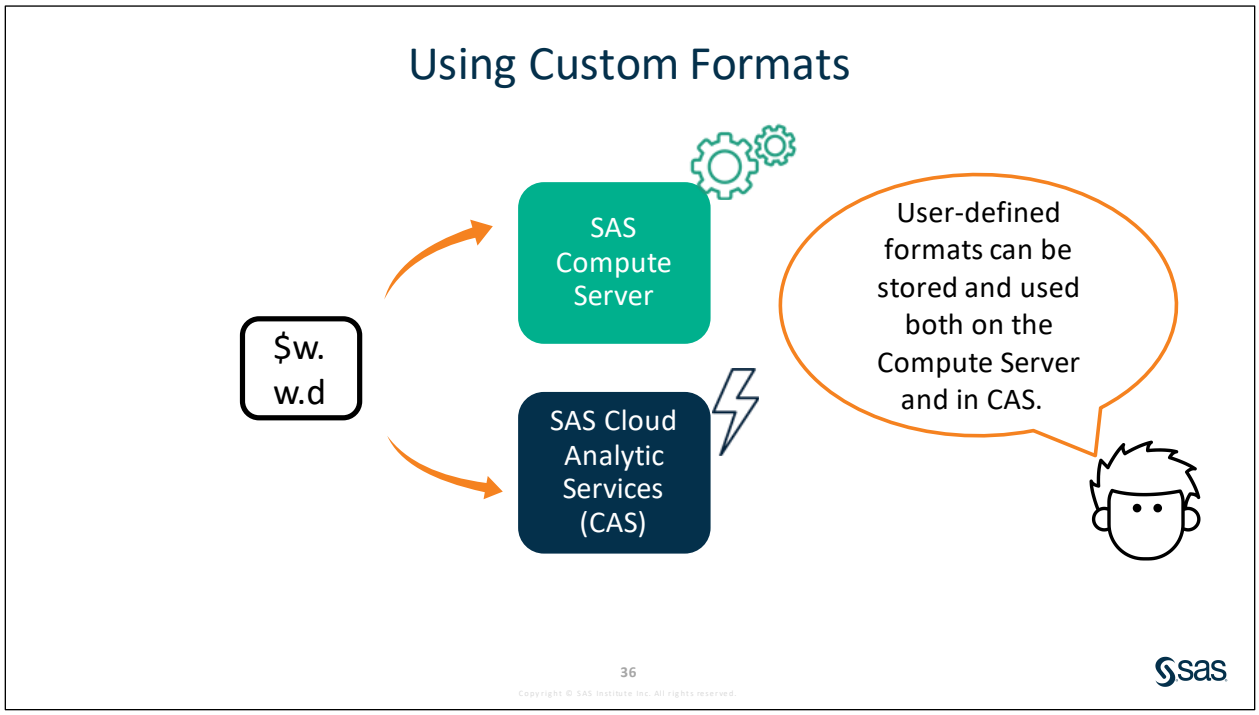
d. PROC CASUTIL can be used to drop the **_NAME_** column from the in-memory table. Use the following syntax for the ALTERTABLE statement:

```
ALTERTABLE CASDATA="table" INCASLIB="caslib" DROP={"col1", "col2"... };
```

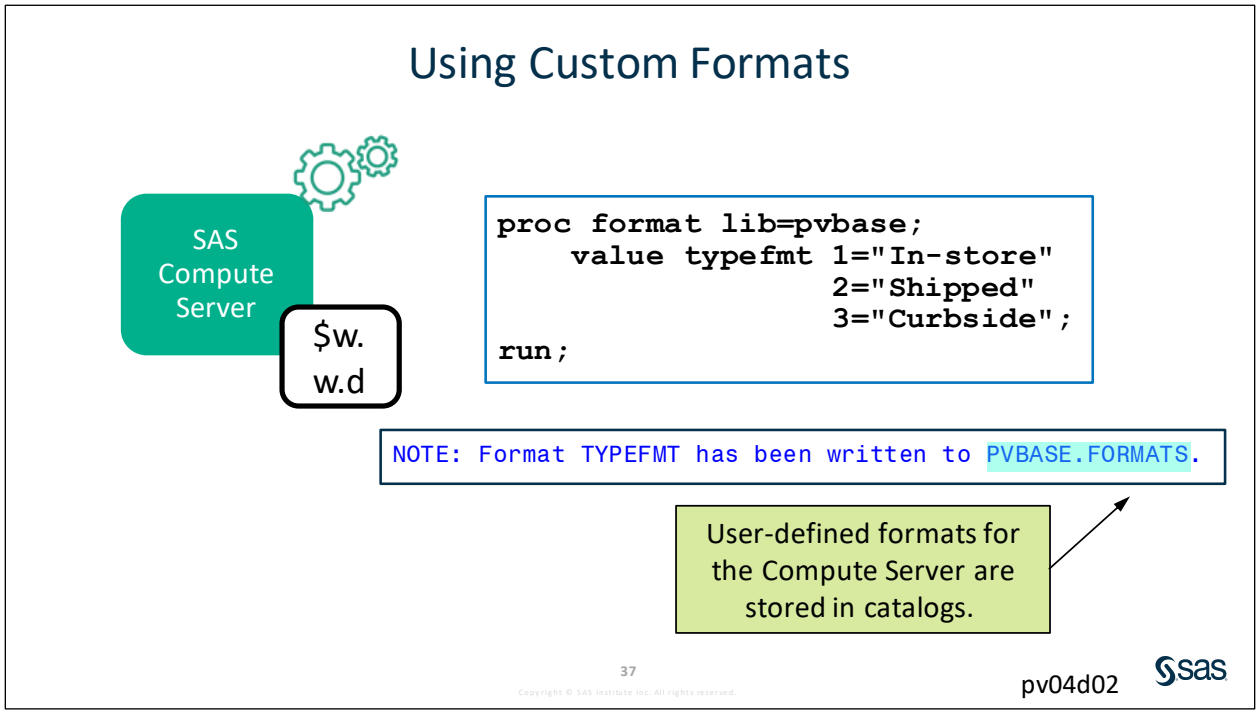
e. Run the program and view the **casuser.qtr_sales_rotate** table. How many rows are included?

End of Practices

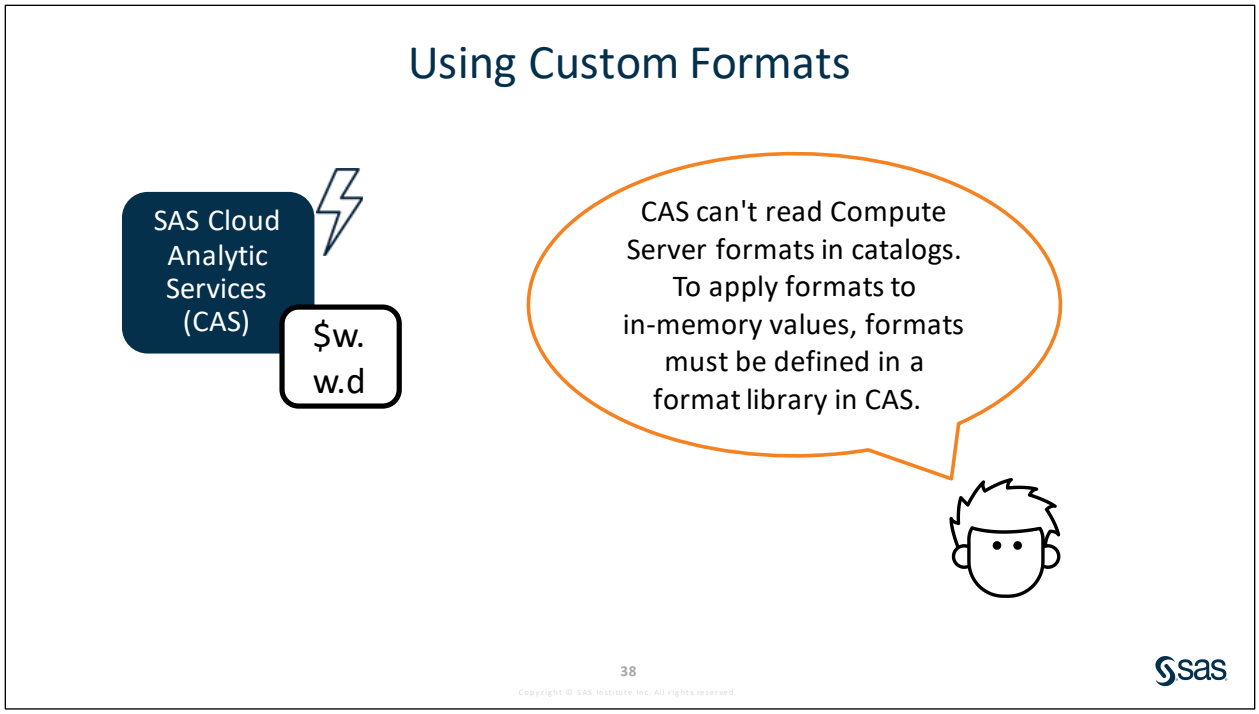
4.2 User-Defined Formats



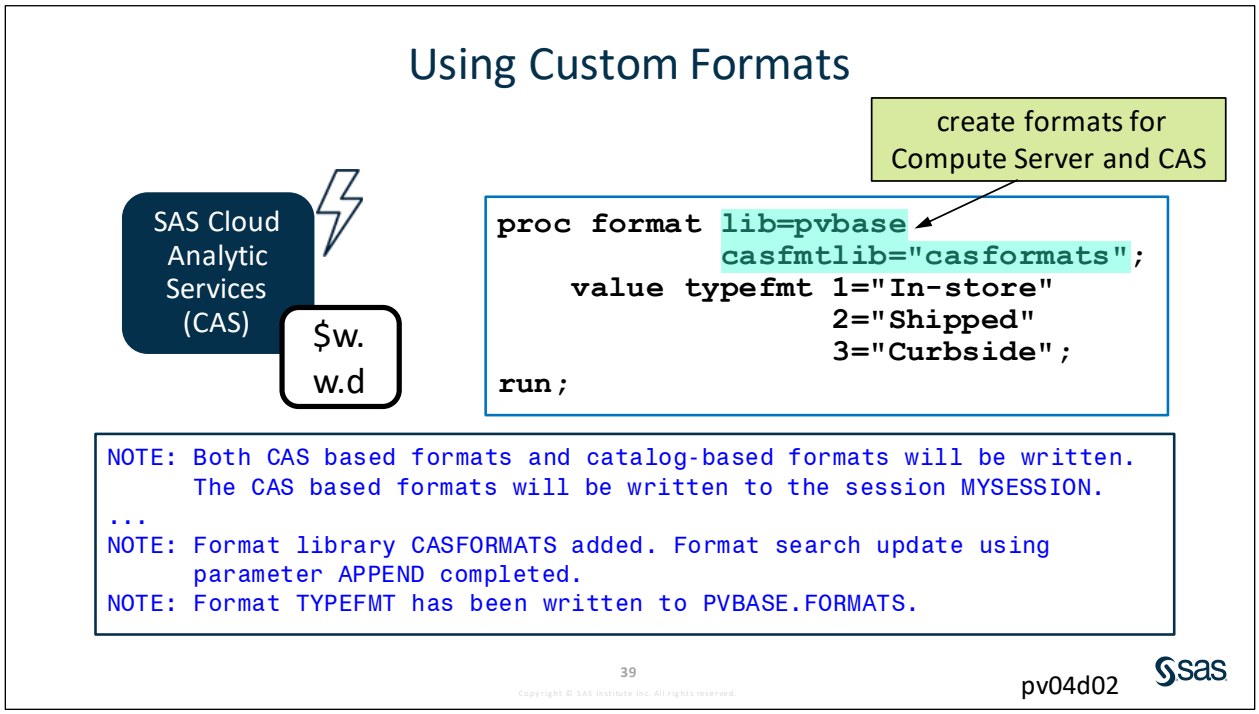
User-defined formats can be stored and used both on the Compute Server and in CAS.



Formats on the Compute Server are stored in catalogs. In this example, the TYPEFMT format is stored in the PVBASE library and FORMATS catalog.



CAS cannot read and apply formats in a Compute Server catalog to in-memory data. Instead, formats in CAS are stored in a format library.



When creating a custom format with PROC FORMAT, you can save it for use both on the Compute Server and in CAS. The LIB= option specifies the library for the Compute Server format, whereas the CASFMTLIB= option names the format library for CAS. The log indicates that the same TYPEFMT format is saved in both locations.

4.03 Activity

Open **pv04a03.sas** from the **activities** folder and perform the following tasks:

1. Highlight and run the CAS statement. The log displays the current CAS format libraries and formats available in the session.
2. Modify the PROC FORMAT statement to add the CASFMTLIB= option. Run the step to create three formats in the **casformats** CAS format library.

```
proc format lib=work.formats casfmtlib="casformats";
```

3. Run the CAS statement again and look at the log. Confirm that **casformats** includes three formats. What is the scope of **casformats**?
4. Run the FREQTAB step. Is the format successfully applied to **OrderType**?

40

Using Custom Formats

assign the format to a column when the data source is loaded into memory

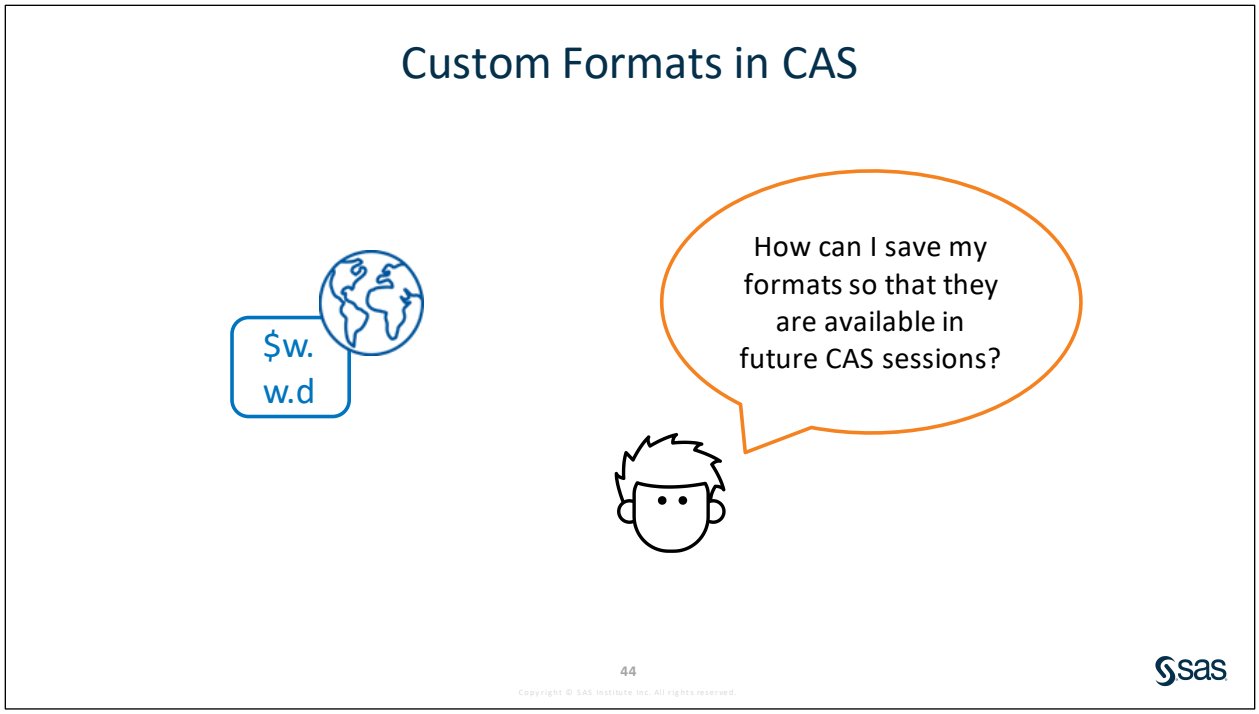
```
proc casutil;
     format OrderType typefmt.;
     load data=pvbase.orders casout="orders"
          outcaslib='casuser' replace;
     contents casdata="orders";
quit;

proc freqtab data=casuser.orders;
     tables OrderType;
run;
```

OrderType	Frequency	Percent	Cumulative Frequency	Cumulative Percent
In-store	715970	75.23	715970	75.23
Shipped	181424	19.06	897394	94.30
Curbside	54275	5.70	951669	100.00

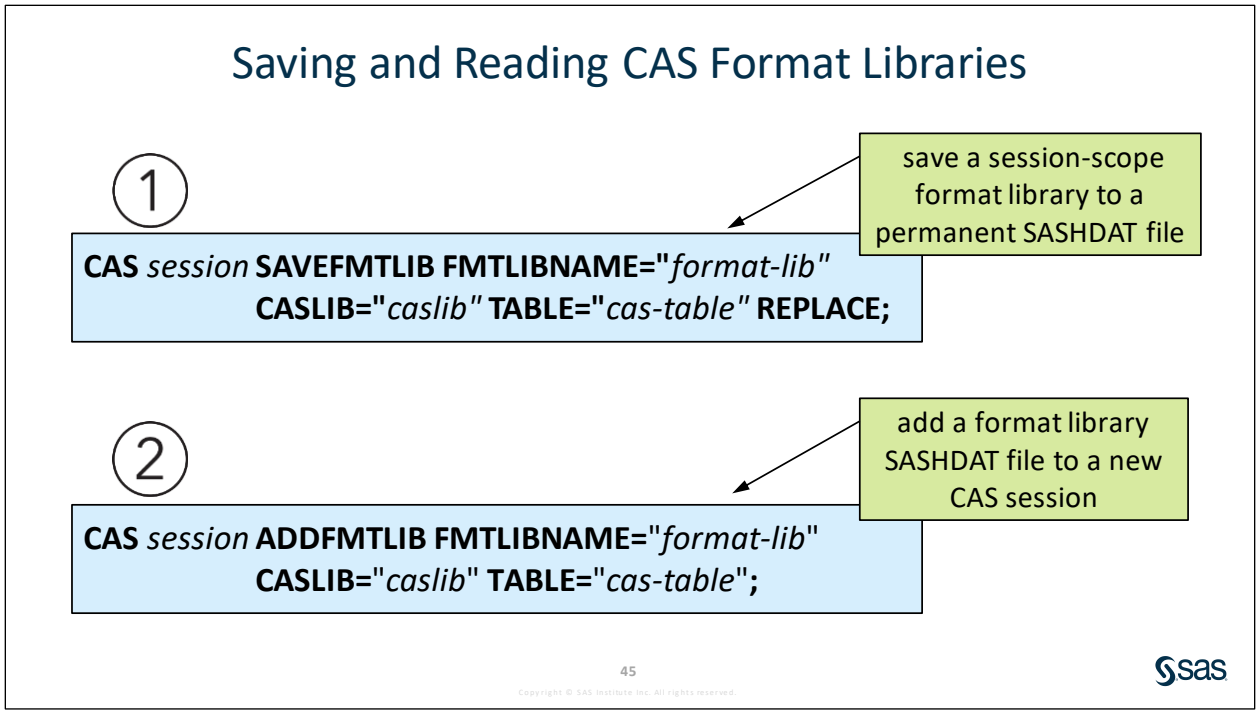
43

pv04d02

For procedures that run on the CAS server, you must assign a format to a variable before the table is loaded onto the server. After a table is in memory, you cannot assign formats to the table variables. To apply formats to table variables before the table is loaded, use a FORMAT statement with the CASUTIL procedure.



How can you save formats so that they are available in future CAS sessions?



The first step is to save the session-scope format library to a permanent SASHDAT file. This is accomplished with the CAS statement and the SAVEFMTLIB option. The CASLIB= option names the caslib where the SASHDAT files should be saved. The TABLE= option specifies the name of the file where the results are stored.

The second step is to configure a new CAS session to read the stored format library. Use the CAS statement with the ADDFMTLIB option. Each of the formats stored in the SASHDAT file will be read and available for use in the new session.

Creating and Using User-Defined Formats in CAS

Scenario

Create a user-defined format in CAS and apply it to an in-memory table. Save the format library as a SASHDAT file.

Program

- **pv04d03.sas**

Syntax

```
PROC FORMAT <LIBRARY=libref<.catalog>
                CASFMTLIB='name'>;
    VALUE <$>name <(format-option(s))> <value-range-set(s)>;
RUN;
```

```
CAS session-name <SAVEFMTLIB FMTLIB=format-library-name <<<TABLE=table-name>
                    <CASLIB=caslib> <REPLACE>> | <PATH=path>> <PROMOTE>>;
```

Notes

- You can specify the CASFMTLIB= option only in an active CAS session.
- User-defined formats defined in the VALUE or PICTURE statements are also written to the format catalog that is specified by the LIBRARY= option.
- If you do not specify a library, then SAS uses the **Work.formats** library.
- In order for a user-defined format to be used in CAS, it has to be created in CAS.
- The user-defined format must be applied to the in-memory table. This can be done by adding a FORMAT statement to the PROC CASUTIL step that loads the table into memory. You can also add a FORMAT statement to a DATA step that references an output CAS table to load the data.
- In order for the user-defined format to be applied within the procedure results, you have to use a procedure that runs in CAS.
- If the procedure has to transfer data to the workspace server to complete processing and the user-defined format has also been defined on the workspace server, the user-defined format that is loaded on the workspace server can be used.
- The SAVEFMTLIB option in the CAS statement saves a session format library to a CAS table or to a file so that it can be used later.

Demo

Open **pv04d03.sas** from the **demos** folder and perform the following tasks:

Note: If you have not set up the autoexec file in SAS Studio, open and submit **startup.sas** first.

1. The PROC FORMAT step creates a user-defined format named RETAILRANGE. The format groups values of **RetailPrice** into five categories.

   Note: The LIB= option saves the format for use on the Compute Server in the **work.formats** catalog. The CASFMTLIB= saves the same format for use in CAS in the **casformats** format library.

2. The CAS statement prints the format definition. Run the PROC FORMAT step and the CAS statement. View the log and confirm that the RETAILRANGE format is created.

```
cas mysession listfmtranges fmtname=retailrange;
```

```
Format Name          Range
RETAILRANGE          LOW-<50=Under $50
                     50-<100=$50-$100
                     100-<200=$100-$200
                     200-<500=$200-$500
                     500-HIGH=Over $500
NOTE: Request to LISTFMTRANGES RETAILRANGE completed for session MYSESSION.
```

3. The PROC CASTIL step performs the following actions:
   a. The DROPTABLE statement drops the **orders** table from memory if it exists in **Casuser**.
   b. The FORMAT statement applies the RETAILRANGE format to the **RetailPrice** column.
   c. The LOAD statement loads **pvbase.orders** to the **orders** in-memory table.
   d. The CONTENTS statement lists table and column attributes for the **orders** table.
   e. Highlight and submit the step. Verify that the RETAILRANGE format is assigned to the **RetailPrice** column.

```
proc casutil;
    droptable casdata="orders" quiet;
    format RetailPrice retailrange.;
    load data=pvbase.orders casout="orders";
    contents casdata="orders";
quit;
```

4. Highlight and submit the PROC FREQTAB step and verify that the custom format was applied to **RetailPrice** in the results.

```
proc freqtab data=casuser.orders;
    tables RetailPrice / nocum;
run;
```

Note: The FORMAT statement is not required in the PROC TABULATE step because the CAS in-memory table was loaded with the RETAILRANGE format applied to the **RetailPrice** column.

The FREQTAB Procedure

Retail Price		
RetailPrice	Frequency	Percent
Under $50	304273	31.97
$50-$100	234320	24.62
$100-$200	222603	23.39
$200-$500	151853	15.96
Over $500	38620	4.06

5. The PROC CASUTIL step saves the **orders** in-memory table with the RETAILRANGE format applied as a SASHDAT file named **orders_format**.

```
proc casutil;
    save casdata="orders" casout="orders_format.sashdat" replace;
quit;
```

6. The CAS statement saves the **casformats** format library into a SASHDAT file so that it can be loaded and used in a subsequent CAS session. Highlight the statement and run the selected code.

   Note: By default, the SASHDAT file will have the same name as the format library. If you would like to specify a different table name, use the TABLE= option.

```
cas mysession savefmtlib fmtlibname="casformats" replace;
```

7. Select **Options** ⇨ **Reset SAS Session**.

8. The remainder of the program includes the following statements and steps:

   a. The CAS statement reads the **casformats.sashdat** table and load formats to the **casformats** format library.

   b. The PROC CASUTIL step loads the **orders_format.sashdat** table to an in-memory table named **orders_format**.

   c. The PROC FREQTAB step generates a report based on the formatted values of **RetailPrice**.

   d. Highlight the remainder of the program and run the selection.

```
cas mysession addfmtlib fmtlibname="casformats";

proc casutil;
    load casdata="orders_format.sashdat" casout="orders_format"
         replace;
quit;

proc freqtab data=casuser.orders_format;
    tables RetailPrice / nocum;

run;
```

End of Demonstration

4.3 Solutions

Solutions to Practices

1. Using the MDSUMMARY Procedure to Generate Statistics in CAS

```
 /* Part a. */
proc casutil;
    droptable casdata="employees" incaslib="casuser" quiet;
    load data=pvbase.employees outcaslib="casuser"
         casout="employees" promote;
quit;

 /* Parts b.-d. */
proc mdsummary data=casuser.employees;
    var Salary;
    output out=casuser.emp_summary;
    groupby Department;
run;

 /* Part e. */
proc mdsummary data=casuser.employees;
    var Salary;
    groupby Department / out=casuser.dept_summary;
    groupby Country / out=casuser.country_summary;
run;

 /* Part f. */
title "Average Salary by Country";
proc print data=casuser.country_summary(drop=Country_f _Column_);
    var Country--_mean_;
    format _M: dollar8.;
run;
title;
```

2. Transposing an In-Memory Table in CAS

```
 /* Part c. */

proc casutil;
    load data=pvbase.qtr_sales outcaslib="casuser"
         casout="qtr_sales";
quit;

proc transpose data=casuser.qtr_sales
               out=casuser.qtr_sales_rotate;
    var Sales;
    id Qtr;
    by Product_ID;
run;


 /* Part d. */

proc casutil;
    altertable casdata="qtr_sales_rotate"
               incaslib="casuser" drop={"_name_"};
quit;
```

e. How many rows are included? **3,151**

End of Solutions

Solutions to Activities and Questions

4.01 Activity – Correct Answer

5. Can the MEANS procedure process in CAS?
   Yes, if the DATA= option specifies a data set with a CAS LIBNAME engine libref.
6. Can the SQL procedure process in CAS?
   No, but FedSQL can process in CAS.

9

sas

continued...

4.02 Activity – Correct Answer

2. What is the name of the CAS action used to perform the summarization?

```
95 proc means data=casuser.orders sum mean;
96 var Cost;
97 class CustomerCountryLabel;
98 run;
NOTE: The CAS aggregation.aggregate action will
be used to perform the initial summarization.
```

PROC MEANS code is converted to a CAS action behind the scenes so that CAS can process the initial summarization.

25

sas

4.02 Activity – Correct Answer

3. Add the MEDIAN option to the PROC MEANS statement and run the program. Does the program run successfully? **No**

```
89 proc means data=casuser.orders sum mean median;
...
ERROR: The maximum allowed bytes (104857600) of data
have been fetched from Cloud Analytic Services. Use the
DATALIMIT option to increase the maximum value.
NOTE: The SAS System stopped processing this step because of errors.
```

The MEDIAN option is not supported in CAS

SAS attempts to download the data for processing on the Compute Server, but the data exceeds the default 100 MB threshold set by the DATALIMIT= option.

26

continued...

4.03 Activity – Correct Answer

3. Confirm that **casformats** includes three formats. What is the scope of **casformats**?

```
82 cas mysession listformats members;
NOTE: Fmtlib = CASFORMATS
Scope = Session
Fmtsearch = YES
Format = $sml
Format = typefmt
Format = ynm
```

CASFORMATS is session scope, which means that the formats are deleted when the CAS session ends.

41

4.03 Activity – Correct Answer

4. Run the FREQTAB step. Is the format successfully applied to **OrderType**?

```
83 proc freqtab data=casuser.orders;
84 format OrderType typefmt.;
ERROR: You cannot apply a format/informat/label to a table
       in Cloud Analytic Services in this procedure. Please
       apply the format/informat/label when the table is created.
85 tables OrderType;
86 run;
```

CAS formats must be applied when data source files are loaded into memory.