

Processing Efficiently in SAS® Viya®

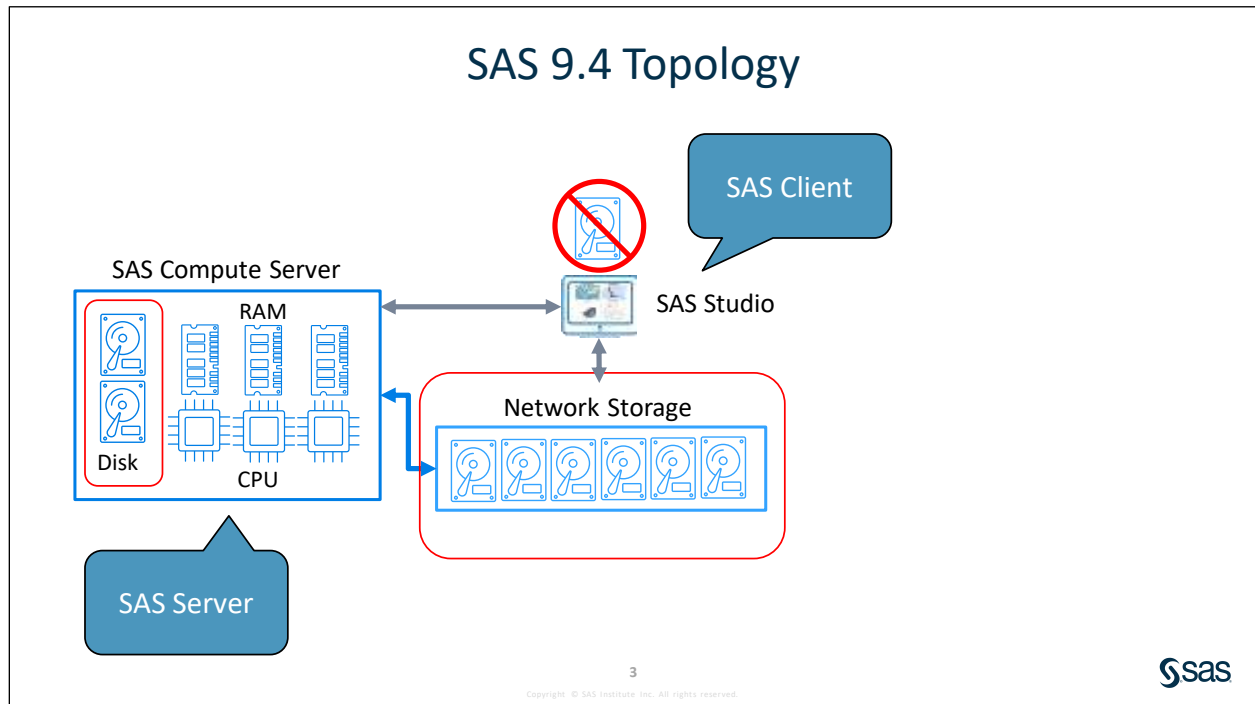
Table of Contents

1.1	Topology	3
1.2	PROC CASUTIL: Managing CAS Data	20
	Demonstration: Demo: Managing CAS Data with PROC CASUTIL	22
1.3	PROC SQL versus PROC FedSQL	24
1.4	Efficiently Working with DBMS Data.....	26
1.5	Joins and Merges	36
	Demonstration: Demo: Managing CAS Data with PROC CASUTIL	37
1.6	PROC CAS	44
	Demonstration: Demo: Executing CASL Code with PROC CAS.....	45
	Demonstration: Demo: Dealing with CASDATALIMIT Issues	48
1.7	Learning More.....	51

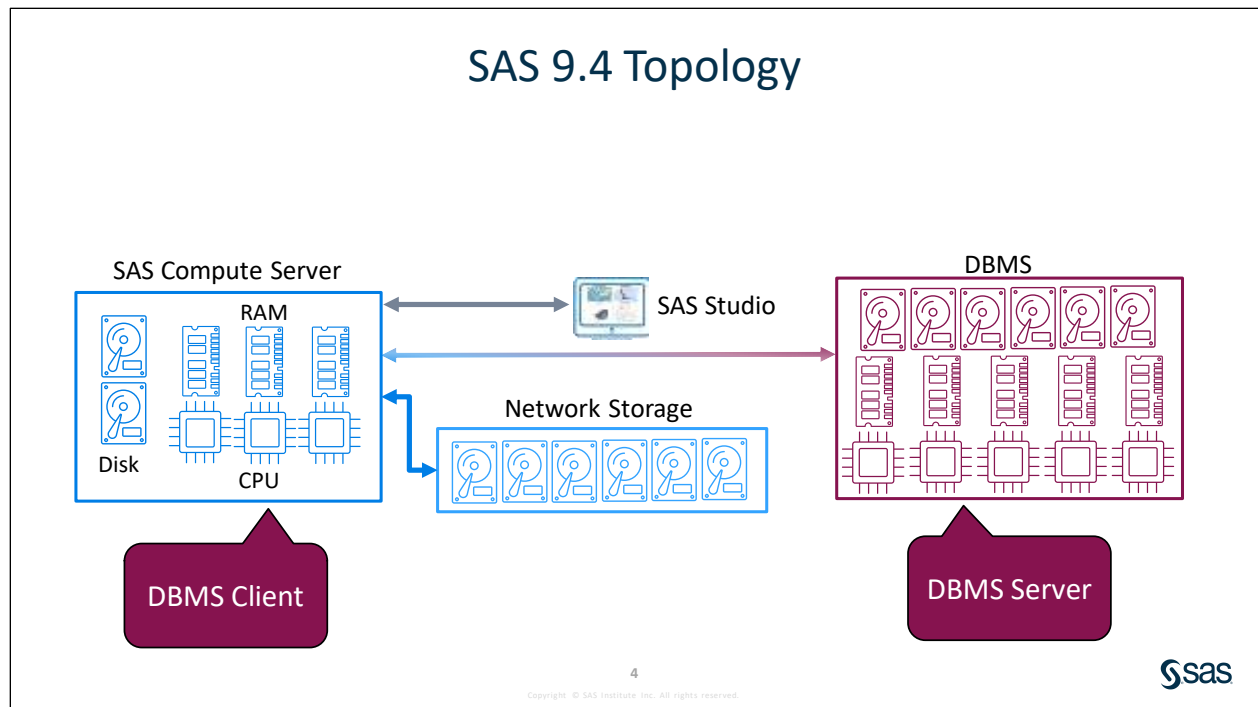
1.1 Topology

“To program effectively in any environment, it is imperative that you understand your system's topology and the effect of that topology on program performance.”

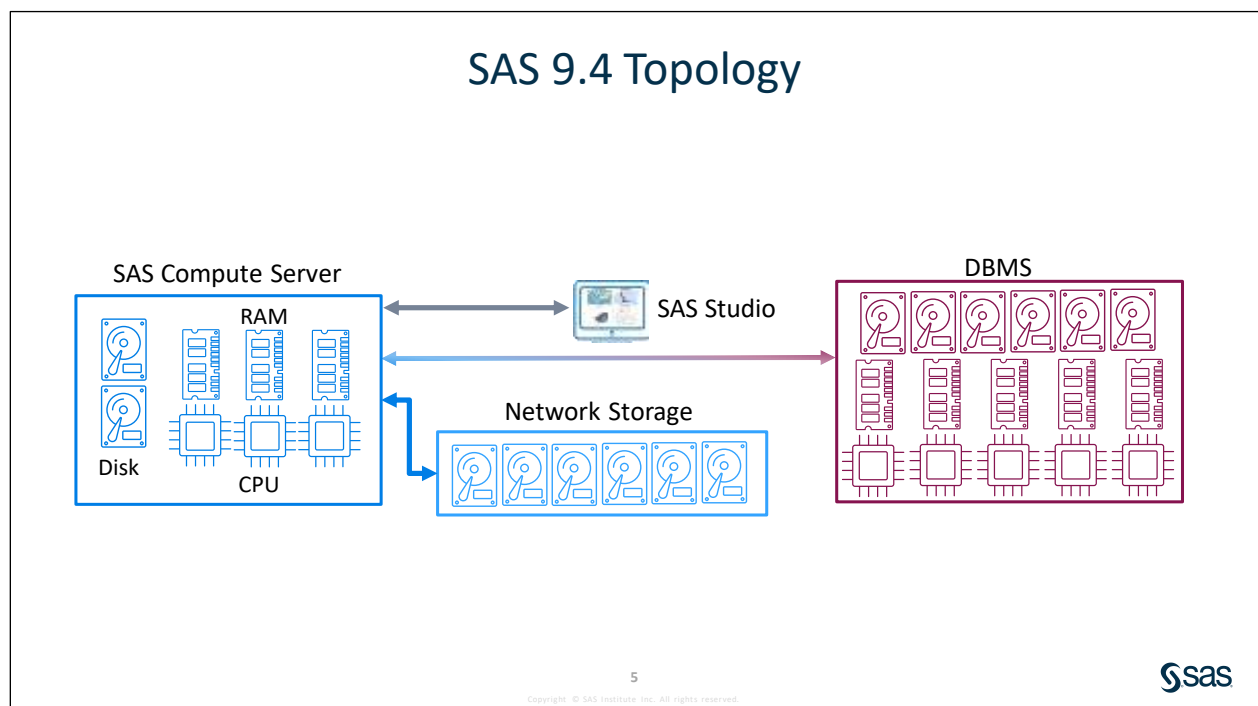
– Mark Jordan

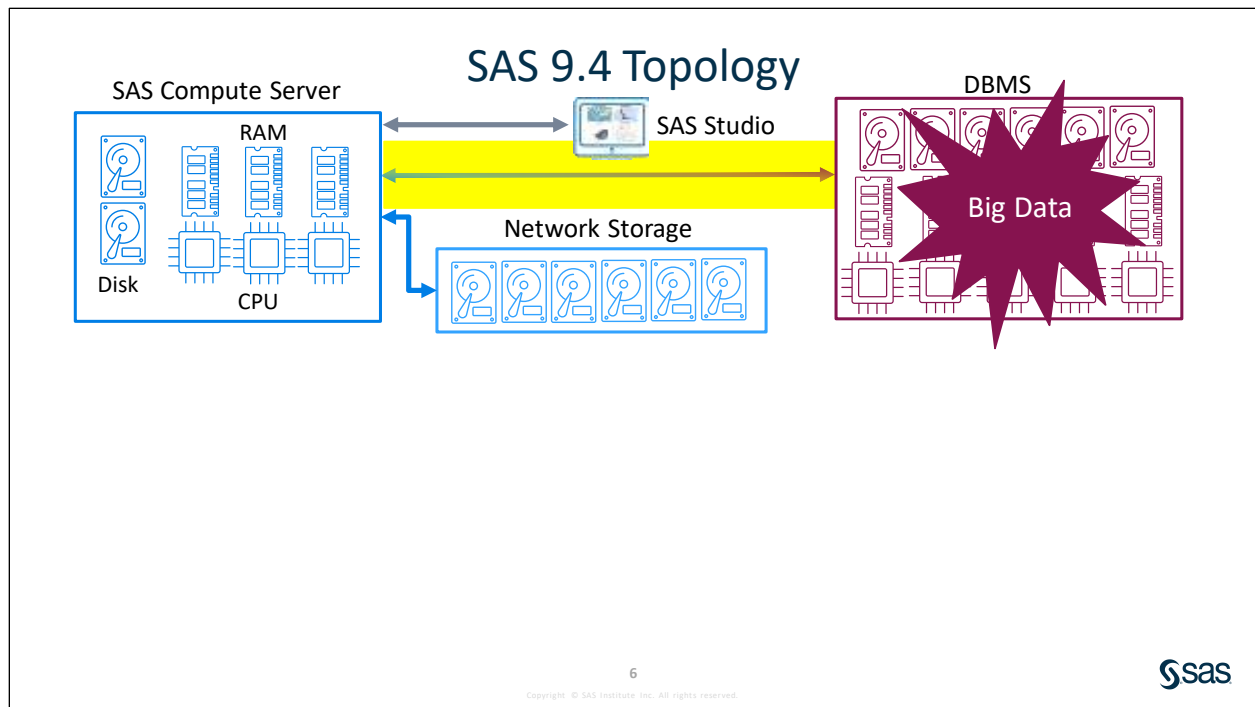


In a typical SAS 9.4 client-server configuration, users access the SAS Compute Server using a SAS client such as SAS Enterprise Guide or SAS Studio. The server portions out CPU and memory resources to each user. SAS data is accessed via the BASE LIBNAME engine, and data is stored on local disks and network volumes available to the Compute Server file system. While the SAS client runs on a user's computer, the Compute Server has no access the local disk on that machine. Files on the SAS client must either be uploaded to the SAS Compute Server or stored in a network location accessible to both the Compute Server and the user's local machine.

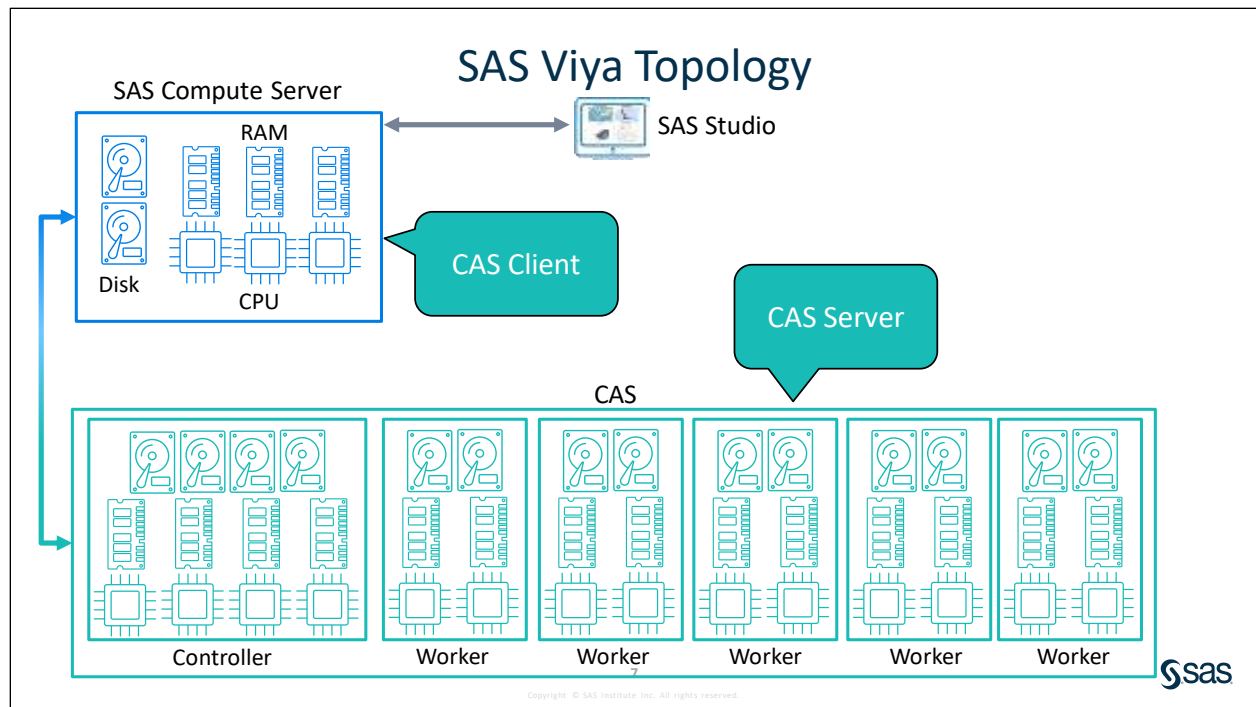


The Compute Server can also include SAS/ACCESS products, enabling LIBNAME access to one or more database management systems, such as Oracle, Teradata, or Athena. When working with DBMS tables, the SAS Compute Server is the client, and the DBMS is the server.

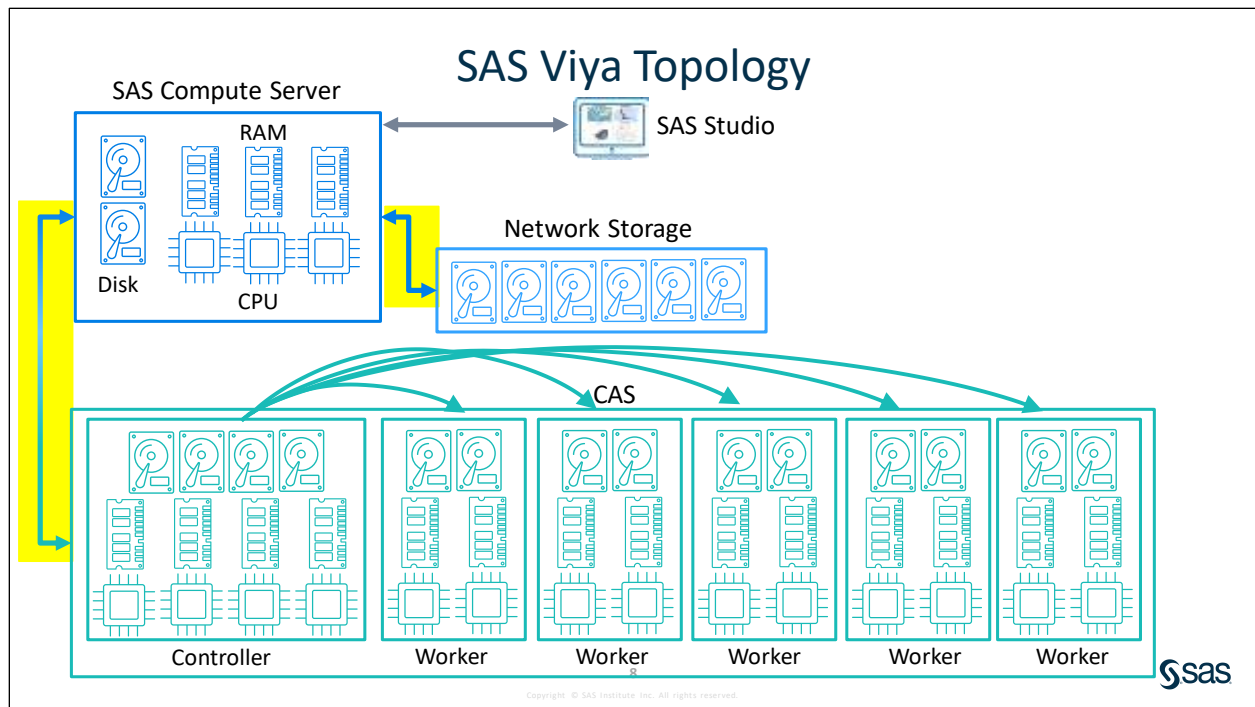




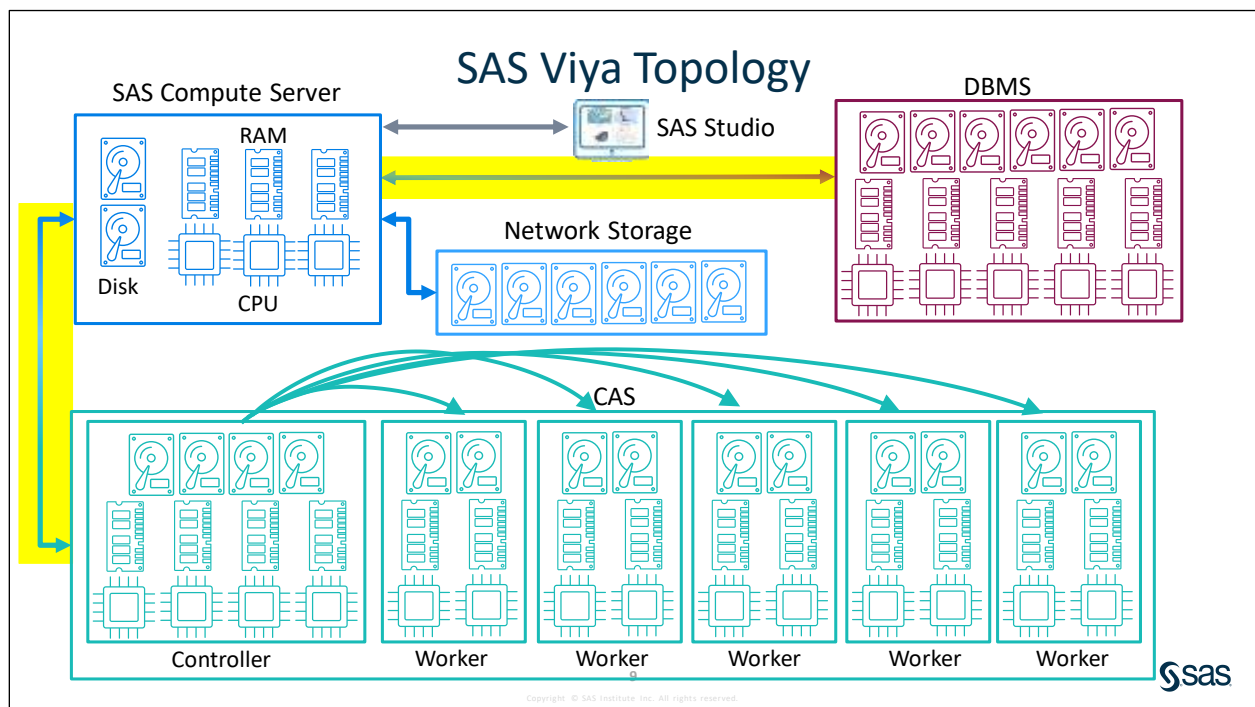
Data in the DBMS can be very large. The DBMS is usually well provisioned with disk, CPU, and RAM, and is optimized for resource-intensive processes such as sorting, subsetting, and summarization. How you write your code and which PROCs you choose to use will significantly affect how much processing happens in the DBMS and how much data must be pulled back to the SAS Compute Server for processing. Maximizing the amount of processing that happens in the DBMS and limiting the amount of data transferred to the Compute Server will make your process run much faster. The goal is to sort, subset, and summarize data in the DBMS, and then take the smaller subset back to the SAS Compute Server where we can use familiar SAS procedures for further analysis, visualization, and reporting.



SAS Viya includes a SAS Compute Server and the Cloud Analytic Service, a massively parallel processing compute engine, commonly referred to as CAS or the CAS server. The Compute Server can pass program code to CAS for processing. In this configuration, the Compute Server is the CAS client, and CAS is the CAS server. CAS offers some distinct advantages when manipulating large data with complex algorithms. CAS tables are preloaded into memory and distributed across the CAS workers. Once loaded, tables persist in memory and can be processed by multiple steps and multiple users without having to be reloaded for each program step. This greatly reduces I/O time. Individual workers use a copy of the program code to process their portion of the table in parallel, significantly speeding up execution of computationally intense programs.

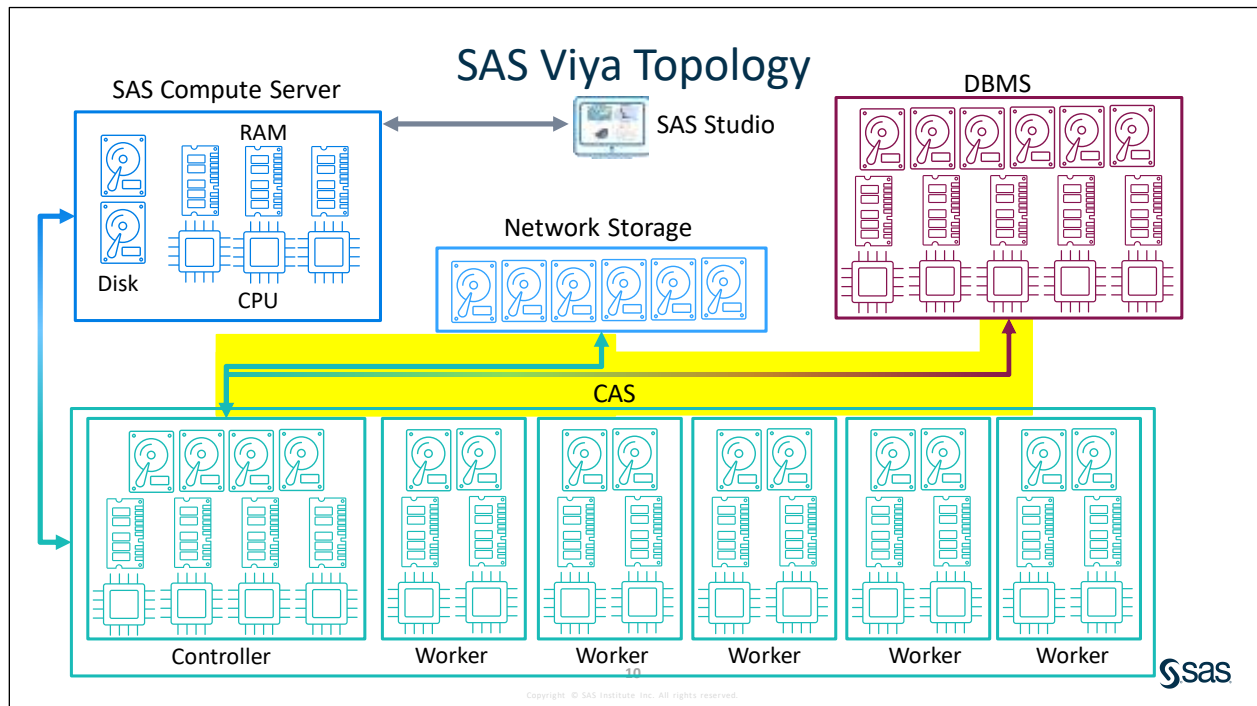


The Compute Server can send code and data to CAS for processing. Data sent to CAS this way must first be accessed on the Compute Server using the LIBNAME engine, then sent to the CAS controller, and then distributed to the CAS workers.

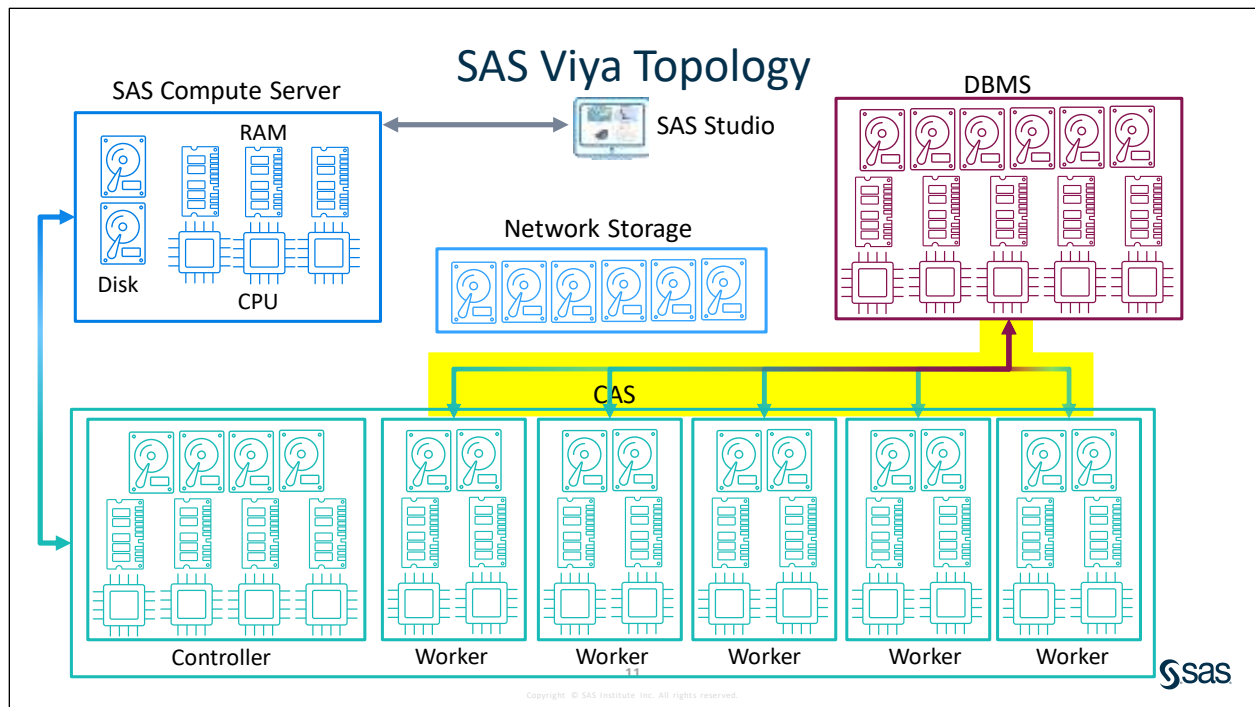


DBMS data accessed via a SAS/ACCESS engine must also be extracted to the Compute Server, and then once again sent to the CAS controller and distributed to the workers. This technique for loading data to CAS is commonly called "client-side loading" because the data must first pass

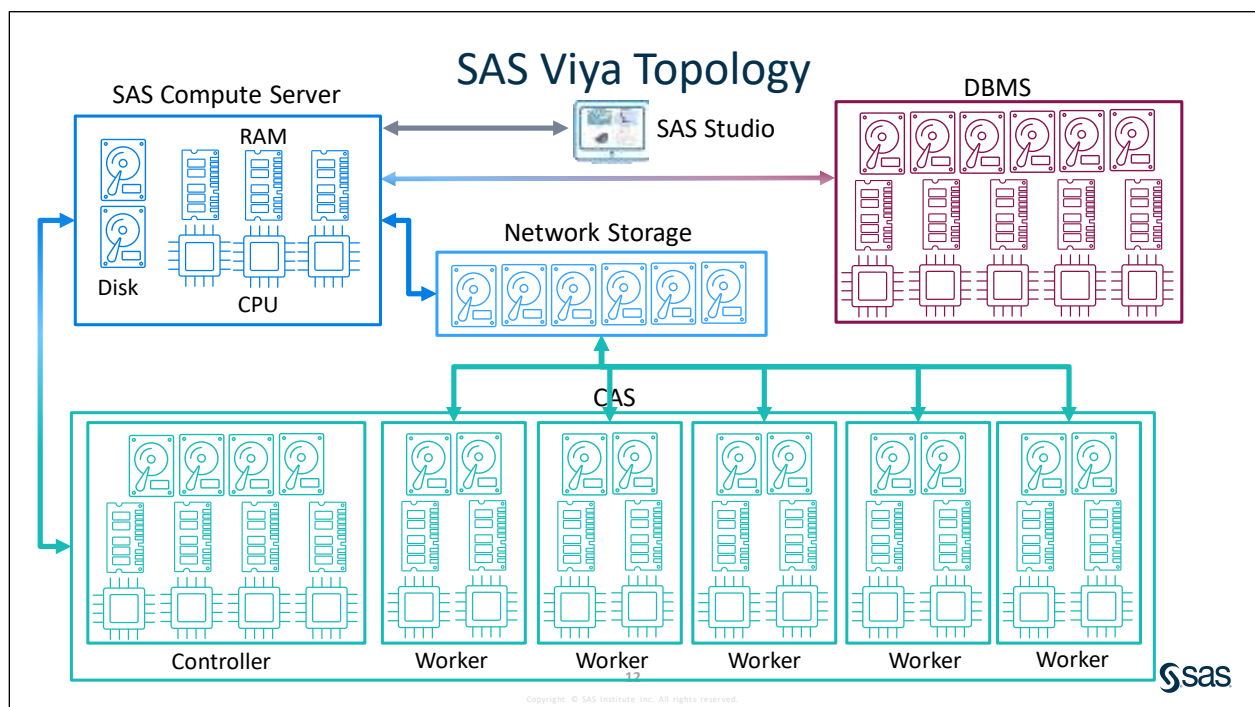
through the client (in this case, the Compute Server) before being passed to CAS. This technique is not optimal for loading large tables into CAS.



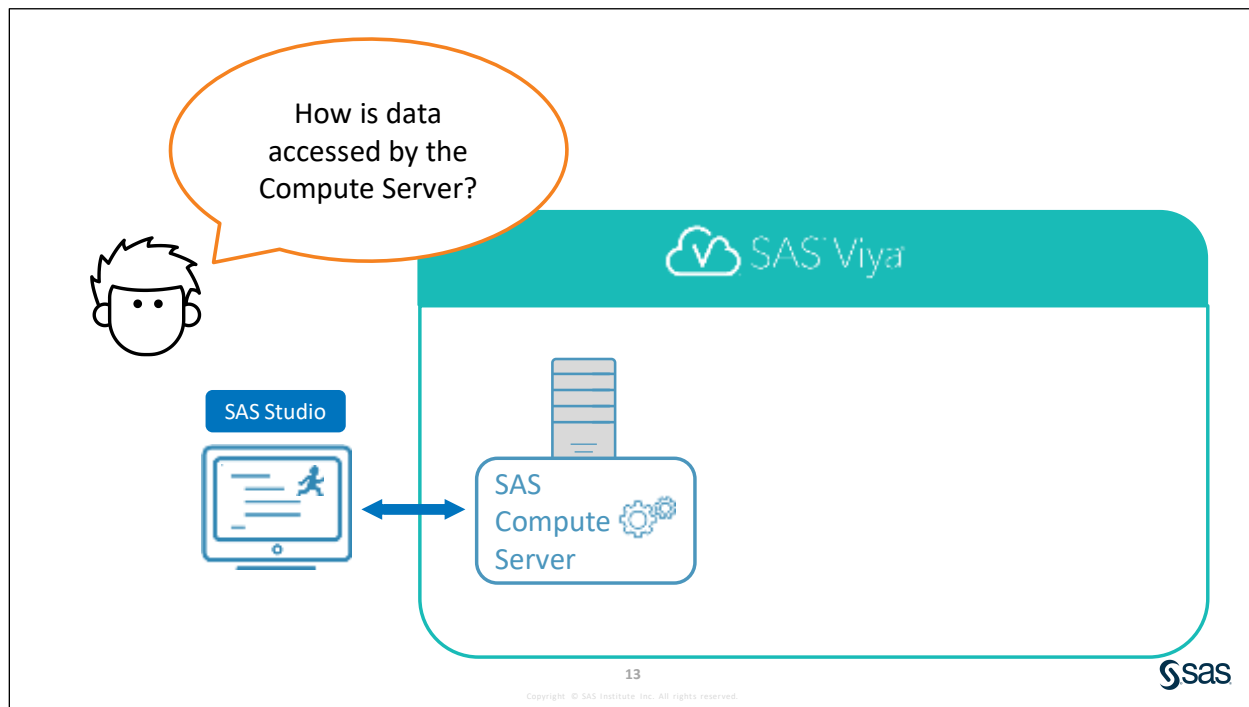
CAS can directly access data from network storage or DBMS systems using technology designed specifically for the MPP environment..



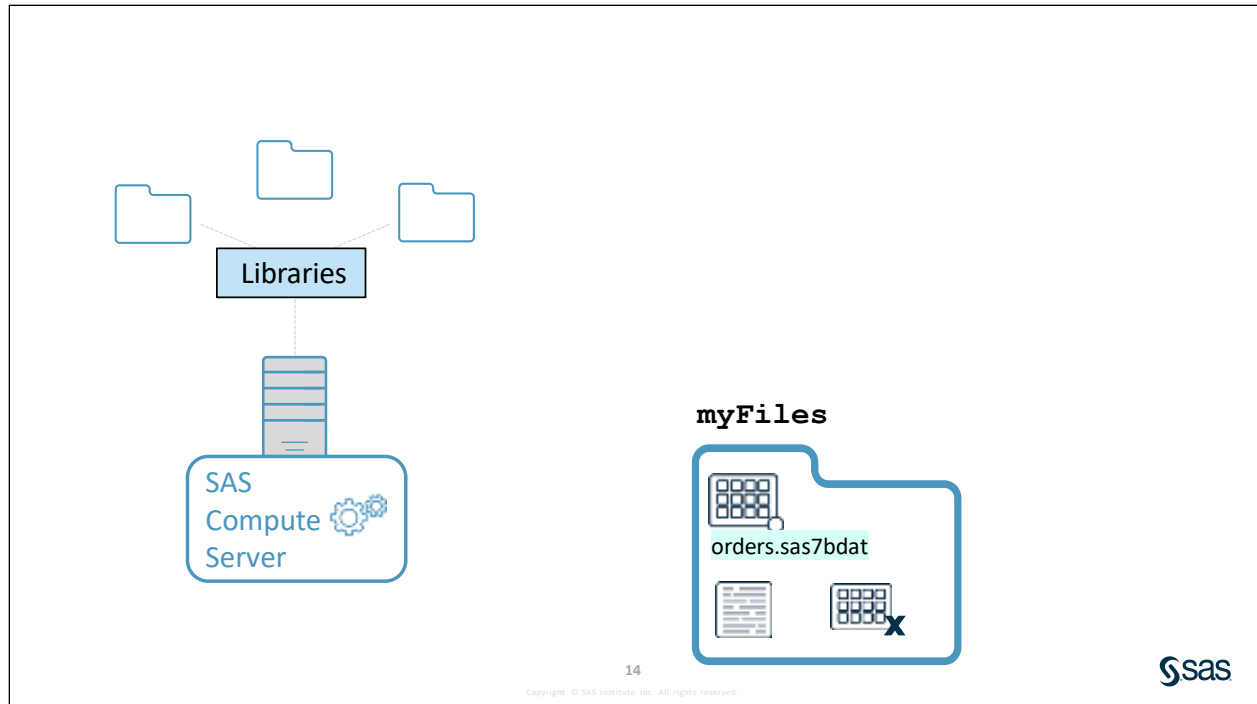
in many cases, this technology enables data to be directly loaded to the workers in parallel, bypassing the controller and dramatically speeding up the loading process. Once loaded in CAS, tables persist in memory, allowing several subsequent processes to work with the data without incurring additional Input/Output (IO) overhead.



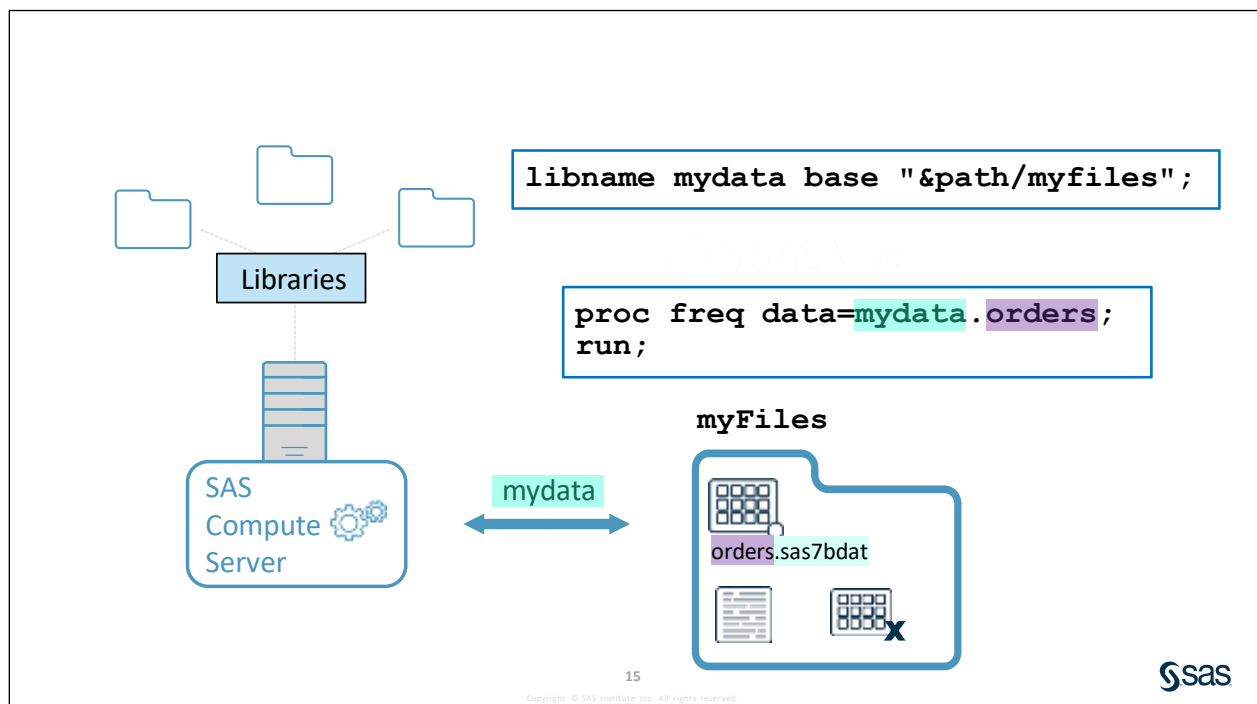
How you write your code and the format in which you store your data can affect the load time for CAS data. How you write your code and which PROCS you choose will affect how much processing happens in CAS and how much data must be pulled back to the Compute Server for processing. The goal is to do the "heavy lifting," like sorting, subsetting, and summarizing, in CAS or the DBMS, and return only small subset or pre-summarized data to the Compute Server for further analysis, visualization, and reporting. Understanding and remaining aware of the topology of your Viya system is the first step toward improving the efficiency of your code.



Let's do a quick review of how data is accessed by the Compute Server.

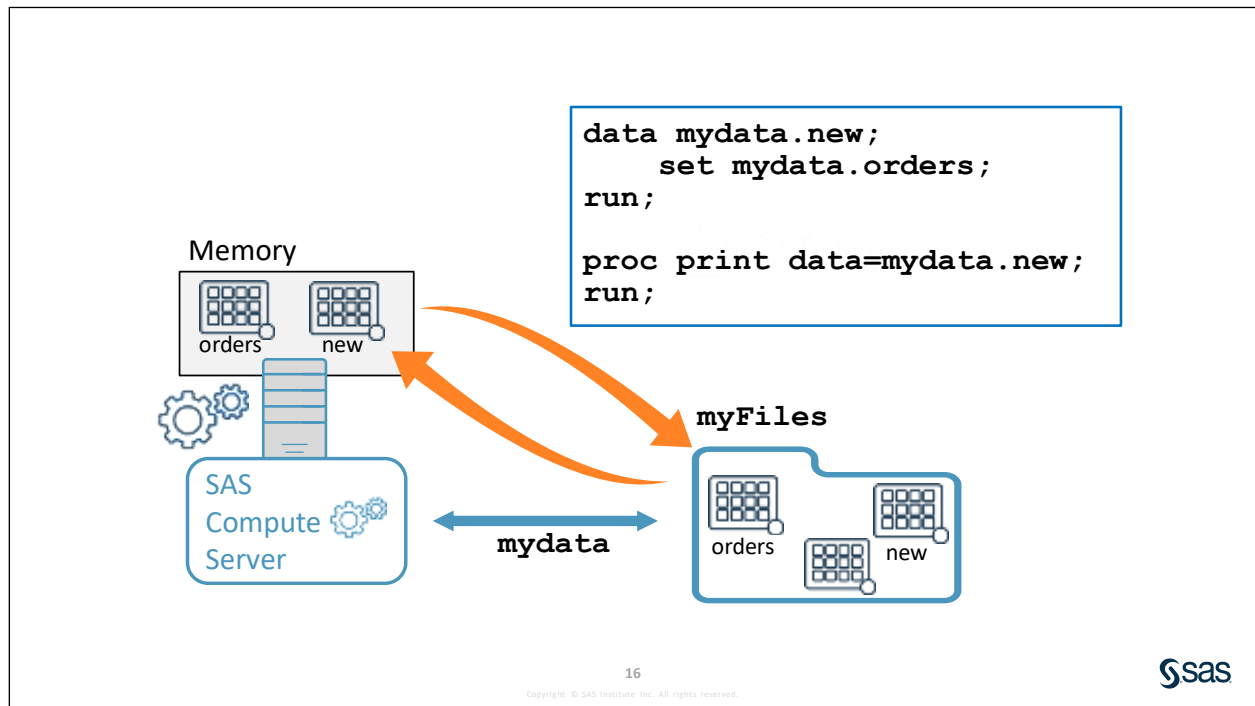


The folder **myFiles** is located in the file system accessible to the Compute Server. It contains several types of files, including a text file, an Excel file, and a SAS data set file named **orders.sas7bdat**.

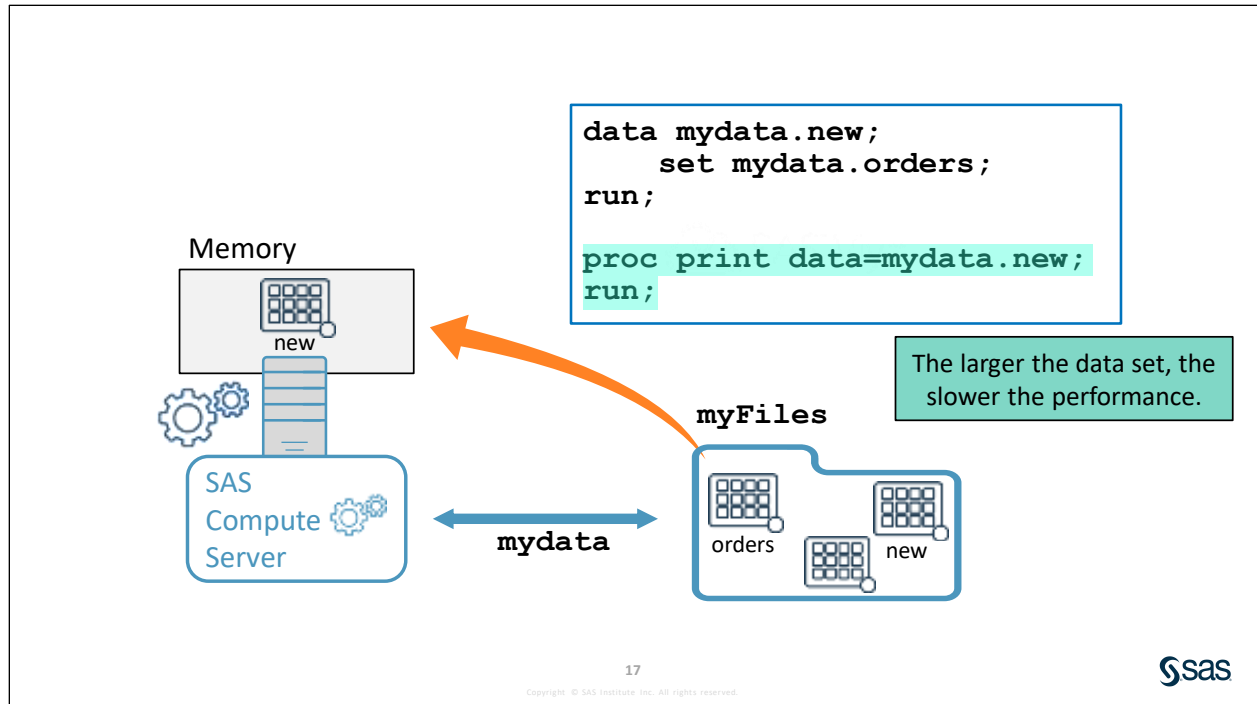


The LIBNAME statement assigns the libref **mydata** to refer to the folder **myFiles**. The BASE engine can access only SAS data in the folder, so the Compute Server sees only the **orders** data

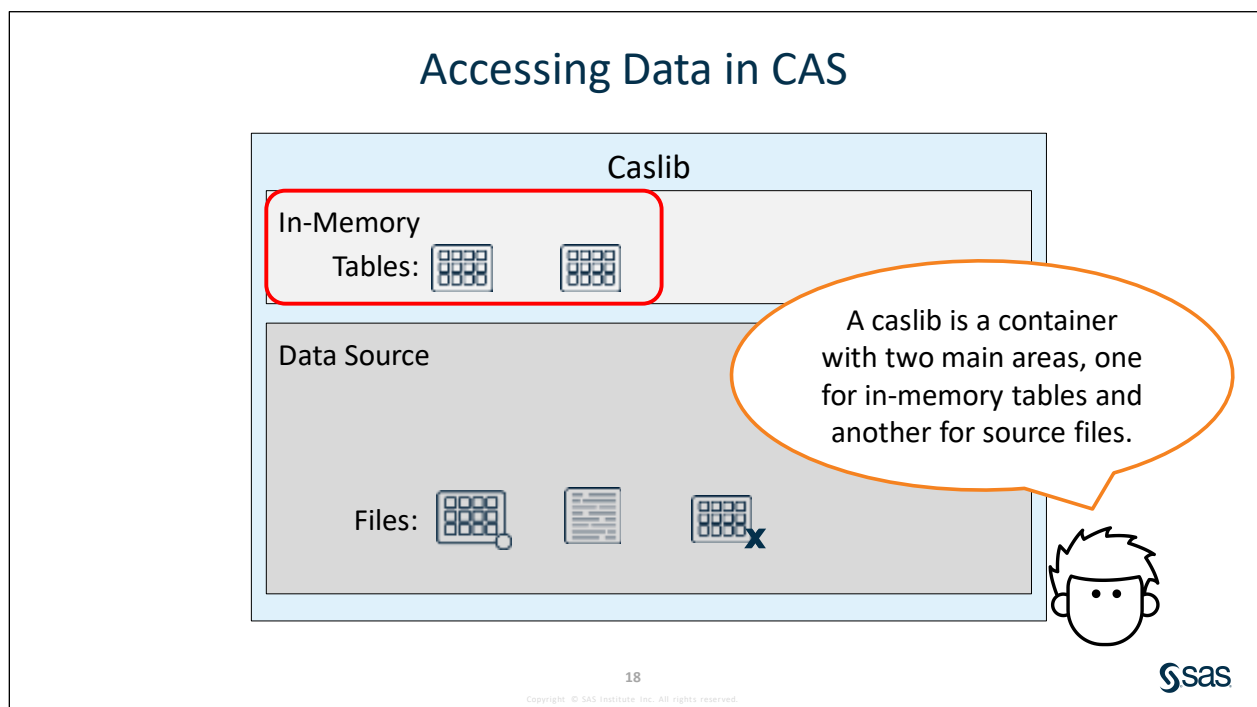
set when using the **mydata** libref. SAS data sets are referenced as *libref.dataset-name*, so this PROC FREQ step reads the **orders** data set in the **mydata** library.



This program reads the **orders** data set, creates a data set named **new**, and then prints the **new** data set. When the DATA step is submitted, the **orders** data set is automatically loaded into the Compute Server's memory, and the DATA step executes, creating the **new** data set. The **new** data set is written to disk, and both data sets are automatically dropped from memory.



When the next step executes, the new data set is loaded back into memory, the PROC PRINT step generates a report, and the **new** data set is again dropped from memory. This process is convenient from the programmer's perspective, but loading and unloading data takes time. The larger the data set, the slower the performance.

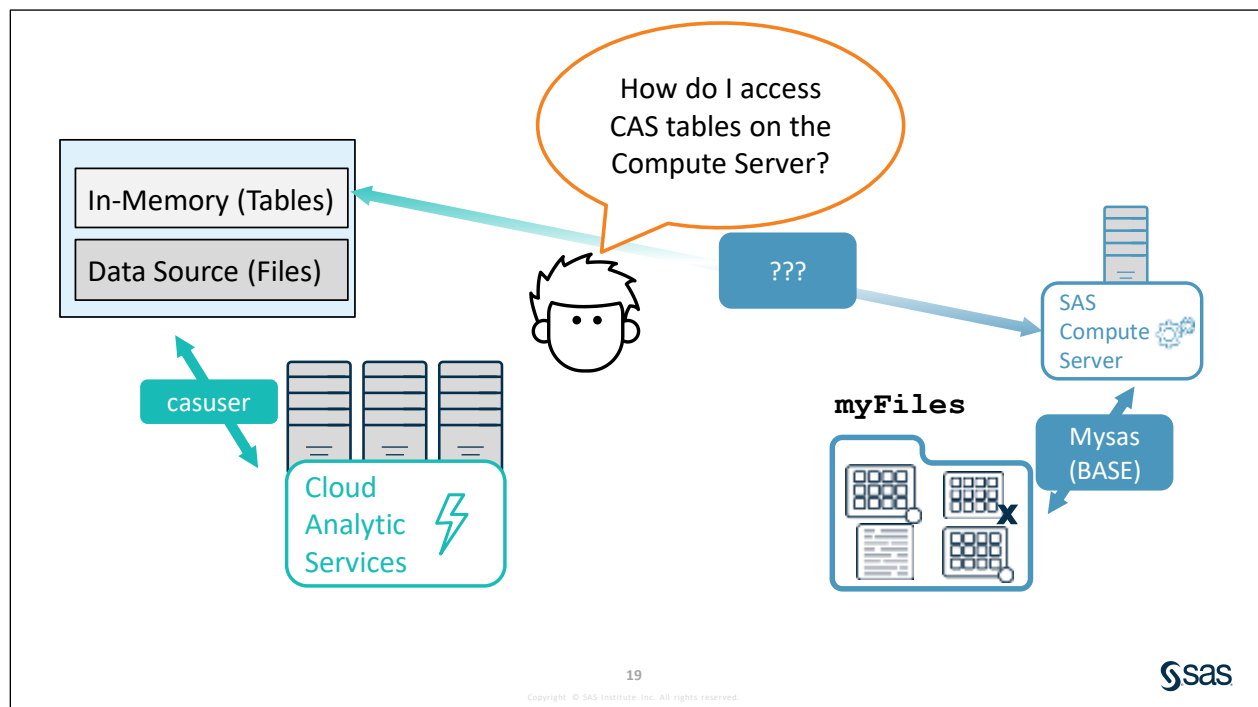


At its simplest, a caslib is a container that has two main areas: one area for in-memory tables and another for data source files.

In CAS, all items stored off-line in the data source are referred to as **files**, no matter what the format.

Only tables loaded into a caslib's memory are referred to as **tables**.

In CAS, only **tables** can be processed.



How can I access CAS tables on the Compute Server?

Adding a Libref for an Existing Caslib

```
LIBNAME libref CAS caslib=caslib-name <sessref=cas-session-name>;
```

- 8 characters or less
- maps the caslib to a libref using the CAS engine
- recommended that the libref and caslib name match if possible

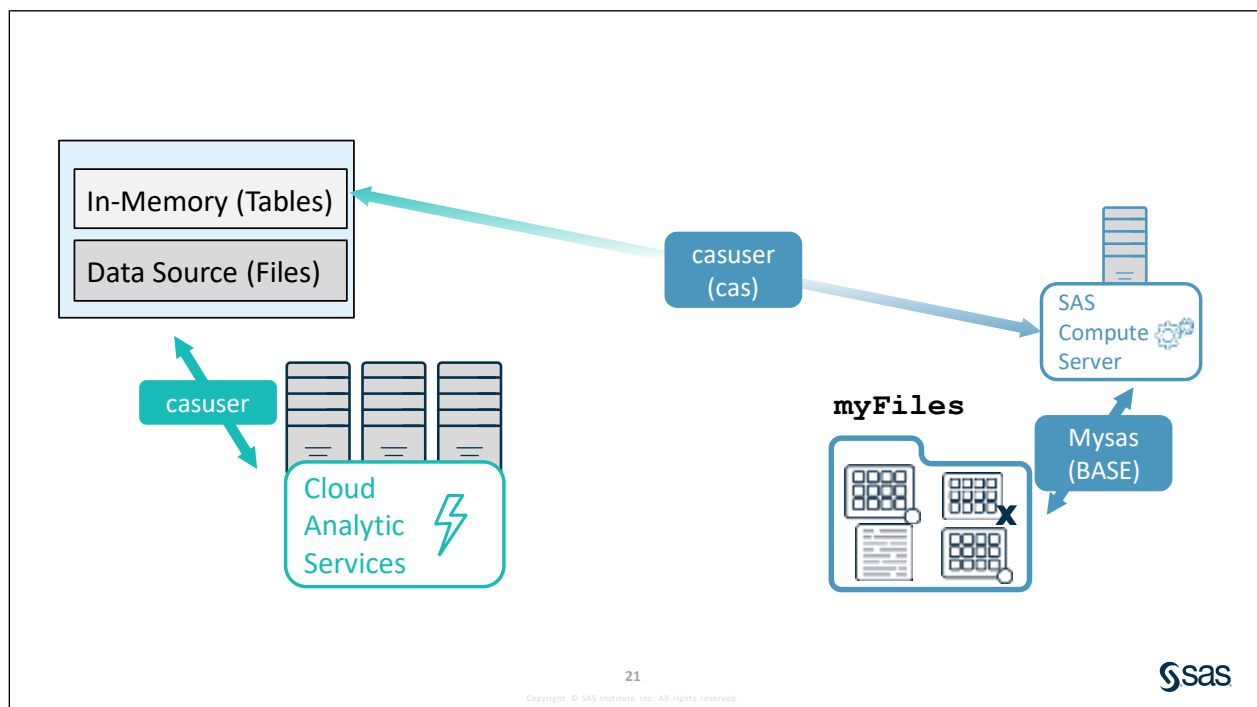
```
libname casuser CAS caslib="casuser" sessref="cs";
```

20

Copyright © SAS Institute Inc. All rights reserved.



To make CAS tables available to the Compute Server, add a libref referencing the existing caslib using the CAS engine. This LIBNAME statement assigns the libref **casuser** to the **casuser** caslib in the CAS session named **cs**.

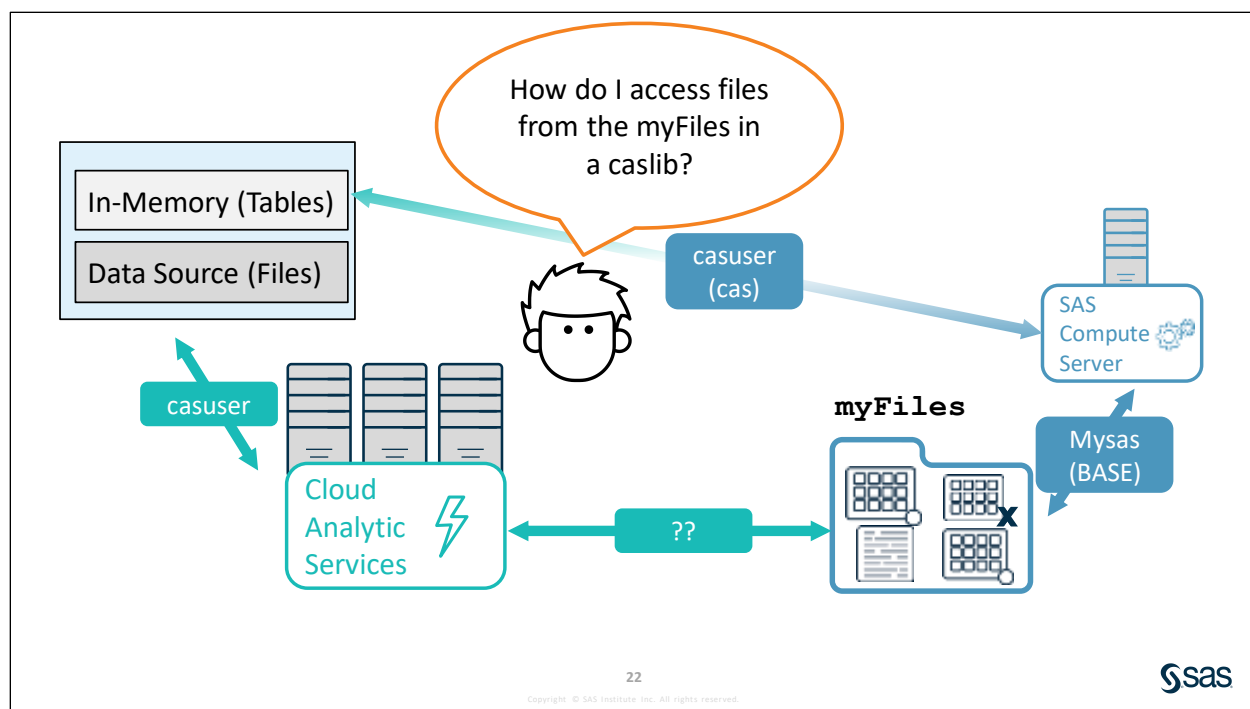


21

Copyright © SAS Institute Inc. All rights reserved.



Any tables loaded in the **casuser** caslib will now be available for Compute Server processing via the CAS engine.



What if I know the path being referenced by a libref on the Compute Server? How can I access those files directly in CAS using a caslib?

Manually Adding a Caslib and Assigning a Libref

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

- 256 characters or less
- can include letters, numbers, and underscores
- cannot start with a number

- 8 characters or less
- maps the caslib to a libref using the CAS engine
- recommended that the libref and caslib name match

```
caslib mysas path="&path/myData" libref=mysas;
```

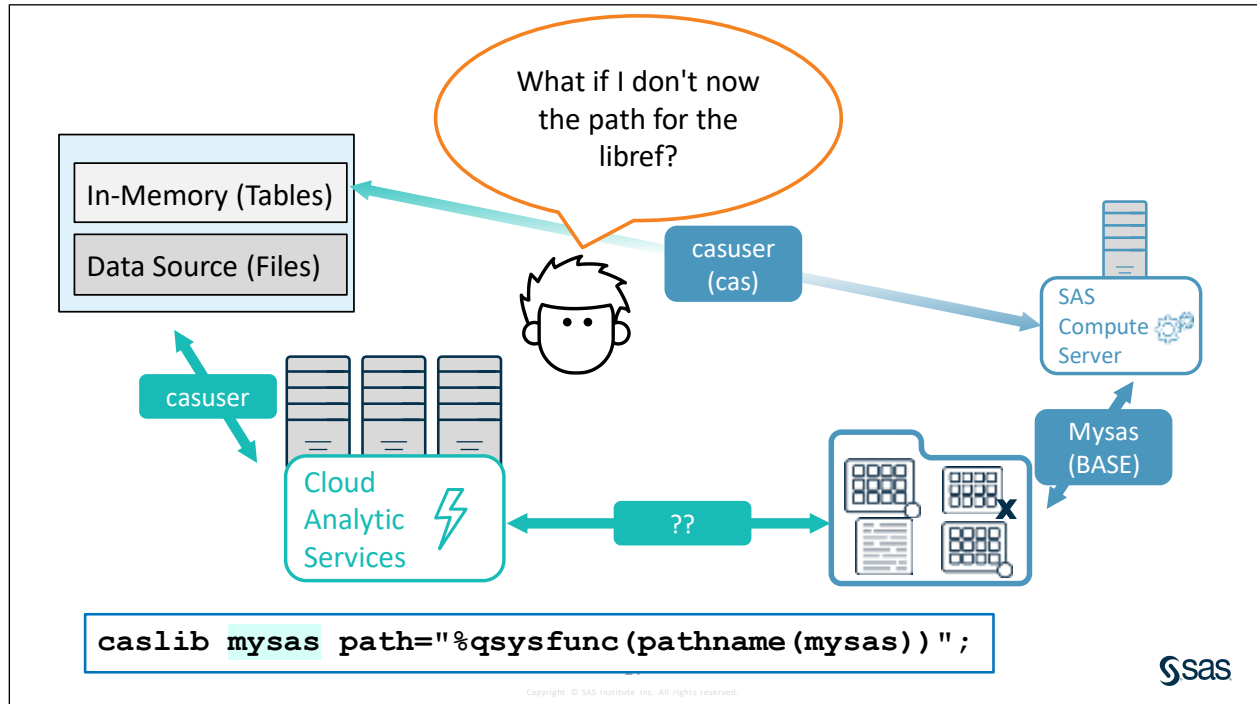
23

Copyright © SAS Institute Inc. All rights reserved.

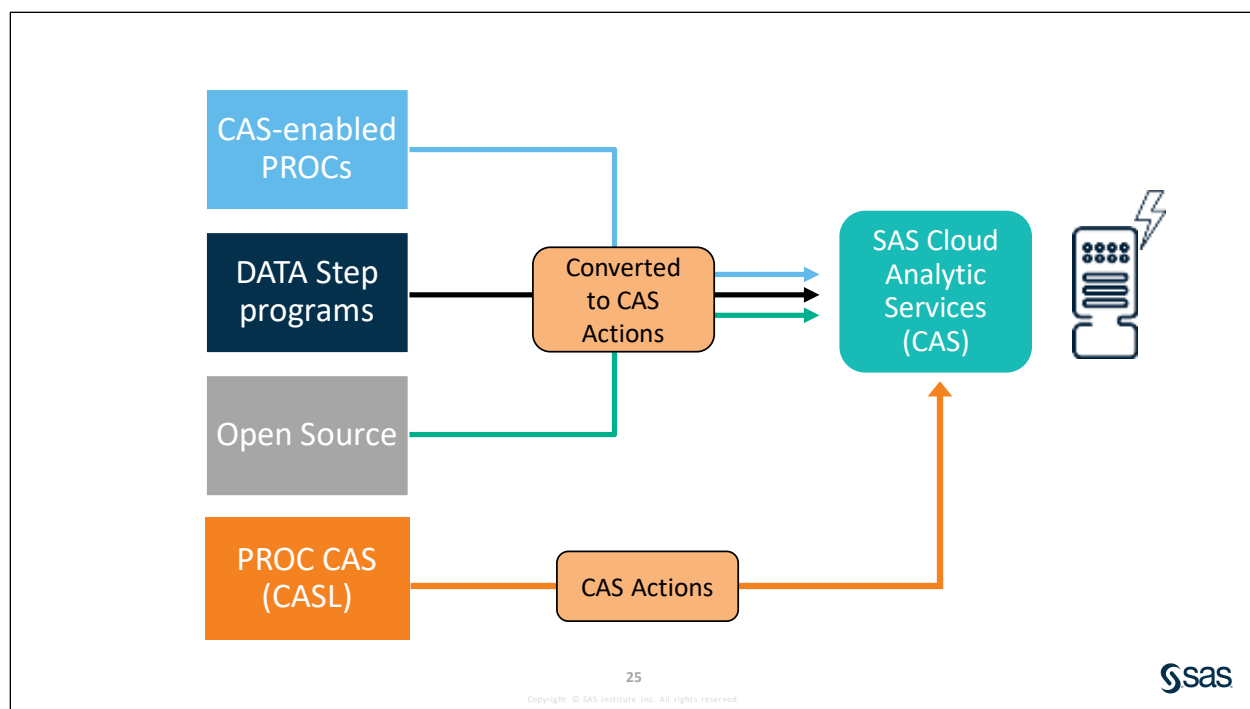
SAS

To manually add a caslib to reference data source files in a folder, you use the CASLIB statement. The caslib name can have up to 256 characters, but otherwise adheres to standard SAS naming rules. The PATH= option specifies the location of the caslib data source files. The LIBREF= option specifies a libref making the caslib tables available to the Compute Server using the CAS engine.

I recommend that the libref match the caslib name whenever possible for clarity when coding. This CASLIB statement adds a caslib named **mysas** to access the data source files in "&path/myData" and assigns a matching libref named **mysas** to point to the caslib.




But what if I only have the libref and not the path information? To assign a caslib to the same folder as an existing SAS library, you can use the SAS macro function %QSYSFUNC along with the PATHNAME function to retrieve path data from a libref. This CASLIB statement adds the caslib named **mysas** to access the data source files in the path referenced by the existing SAS libref **mysas**. Files in the **mysas** caslib are now accessible to CAS as server-side files, and to the Compute Server using the standard BASE engine.





When you have access to SAS Viya, there are multiple ways to interact with the CAS server to process data. You can use CAS-enabled PROCs, traditional DATA step programs, or open-source languages. All of these techniques leverage the CAS API to convert native language elements to CAS actions for execution in CAS. From the Compute Server, you can also use PROC CAS to write CASL code that can specify CAS actions for direct processing in CAS with much more control. With access to both the Compute Server and CAS, it's good to understand the benefits and considerations for each.


Compute Server or CAS?




Which program steps
should run better in
CAS than the
Compute Server?

- 

steps using data sources
larger than 50 GB
- 


long-running steps
(30 minutes or longer)
- 

computationally demanding PROCs
- 

DATA steps with many computations,
functions, or conditional logic

26

Copyright © SAS Institute Inc. All rights reserved.



The Compute Server is a familiar, mature, and efficient processing environment. But CAS will usually outperform the Compute Server with data sources larger than 50 GB, program steps that run for 30 minutes or longer, and computationally demanding programs using complex logic. If you have existing programs that run well on the Compute Server, good – let 'em run! But if performance needs improvement or you are writing new code and any of these conditions apply, consider writing or rewriting it for execution in CAS.

1.2 PROC CASUTIL: Managing CAS Data

CASUTIL Procedure

```
PROC CASUTIL <INCASLIB="caslib-name">;
  LIST FILES|TABLES <option(s)>;
  LOAD DATA=|CASDATA= INCASLIB= CASOUT= OUTCASLIB=;
  CONTENTS  CASDATA= INCASLIB=;
  ALTERNATE CASDATA= INCASLIB= <options>;
  SAVE DATA=|CASDATA= INCASLIB= CASOUT= OUTCASLIB=;
  DROPTABLE CASDATA= INCASLIB= <options>;
  DELETESOURCE CASDATA= INCASLIB= <options>;
QUIT;
```

29

Copyright © SAS Institute Inc. All rights reserved.



We can use the CASUTIL procedure to manage caslibs, data source files, and in-memory tables using syntax that is more familiar to traditional SAS programmers. The code that we write with PROC CAUTIL is converted into CAS actions and sent to CAS for processing. Let's look at a demonstration of using PROC CASUTIL to interact with CAS.

Just for documentation purposes:

The INCASLIB= option can be used to specify a default caslib for the procedure, if you don't want to write the "INCASLIB=" option values in each statement.

The LIST statement lists either the data source files or the in-memory tables.

The LOAD statement loads a table to the caslib. The source can be a SAS data set or a file from any caslib's data source.

The CONTENTS statement is used to get information about a CAS table.

The ALTERNATE statement is used to modify the table name and column metadata.

The SAVE statement is used to save a copy of an in-memory CAS table to a data source file for offline storage.

The DROPTABLE statement removes a table from memory. Tables are **not** automatically saved before dropping.

The DELETESOURCE statements is used to delete a file from a caslib's data source.



Demo: Managing CAS Data with PROC CASUTIL

Program file: **proc casutil.sas**

```

/*****
PROC CASUTIL:
  Requires a CAS session
  Runs on the Compute Server, generates CASL that executes in CAS
  Perform the following actions:
    - List the in-memory tables already loaded in a caslib.
    - List the files in a caslib data source.
    - Load files from the caslib data source to in-memory tables.
    - View table column names, data types, and other column
information.
    - Alter table name, column formats, etc.
    - Save CAS tables as files in the caslib data source.
*****/

proc casutil ;
list tables incaslib="casuser";
run;

proc casutil ;
list files incaslib="casuser";
run;

/* No tables in db2cas */
proc casutil ;
list tables incaslib="db2cas";
run;

proc casutil ;
list files incaslib="db2cas";
run;

proc casutil ;
list files incaslib="db2cas" DATASOURCEOPTIONS=(SCHEMA="STUDENT");
run;

proc casutil;
  load data=sashelp.cars (where=(Make="Toyota")) casout="Toyotas"
outcaslib="casuser" replace;
  save casdata="Toyotas" incaslib="casuser"
casout="toyotas.csv" outcaslib="casuser" replace;
  list files incaslib="casuser";
  list tables incaslib="casuser";
  contents casdata="toyotas" incaslib="casuser";
run;

```

```

proc casutil;
  contents casdata="toyotas" incaslib="casuser";
  altertable casdata="Toyotas" incaslib="casuser"
    /* rename="ToyotaCars" */
    drop={"Origin", 'Type', "Length", "Weight", "Wheelbase"};
  contents casdata="Toyotas" incaslib="casuser";
run;

proc casutil;
  droptable casdata="Toyotas" incaslib="casuser";
  list files incaslib="casuser";
  list tables incaslib="casuser";
run;

proc casutil;
  load casdata="toyotas.csv" incaslib="casuser" casout="Toyotas"
  outcaslib="casuser" replace;
  contents casdata="Toyotas" incaslib="casuser";
run;

proc casutil;
  altertable casdata="Toyotas" incaslib="casuser"
    rename="ToyotaCars"
  COLUMNS={ {NAME="MSRP", FORMAT="COMMA12.2"}, {NAME="INVOICE", FORMAT="COMMA12.2"} };
run;

proc casutil;
  load casdata="toyotas.csv" incaslib="casuser" casout="Toyotas"
  outcaslib="casuser" replace;
  contents casdata="ToyotaCars" incaslib="casuser";
  contents casdata="Toyotas" incaslib="casuser";
run;

proc casutil;
  droptable casdata="Toyotas" incaslib="casuser";
  droptable casdata="ToyotaCars" incaslib="casuser";
run;

```

End of Demonstration

1.3 PROC SQL versus PROC FedSQL

PROC SQL vs. PROC FedSQL

PROC SQL

- SAS proprietary implementation of ANSI SQL 2 (1992) specification
- Can process only CHAR and DOUBLE data types
- Includes many non-ANSI SAS enhancements
- May require significant re-textualization to execute in-database
- Single-threaded except sort/index
- Cannot process in CAS

PROC FedSQL

- SAS proprietary implementation of ANSI SQL 3 (1999) specification
- Processes 17 ANSI data types
- Includes very few non-ANSI SAS enhancements
- Vendor-neutral syntax is easily executed in-database
- Fully multi-threaded
- Processes natively in CAS

32

Copyright © SAS Institute Inc. All rights reserved.



PROC SQL versus PROC FedSQL – Programmer Notes

PROC SQL

Use standard operators or mnemonics (NE, GT, etc.)

CALCULATED keyword

REMERGE

SELECT column Format= and Label=

Can access data in any SAS libref

Can access DBMS data via any SAS/ACCESS engine

PROC FedSQL

Only standard operators (<>, >, etc.)

Re-specify the expression

Use subquery and a join

Not supported

Concatenated SAS librefs not supported

Can access DBMS data when a threaded driver is available

Copyright © SAS Institute Inc. All rights reserved.



There are significant differences between PROC SQL and PROC FedSQL, which adheres more closely to the ANSI standards. You can use standard operators or their mnemonic equivalents in PROC SQL queries, but only the standard operators in PROC FedSQL. The CALCULATED keyword is a PROC SQL SAS enhancement not available in PROC FedSQL. Instead, PROC FedSQL uses the

ANSI standard technique of rewriting the expression that produced the computed column's value. For queries that request both row-level details and summarized values, PROC SQL automatically calculates the summarized values and remerges the result with individual rows, but PROC FedSQL throws an error and stops processing just as an ANSI DBMS would. You must explicitly code a summary subquery and then join the result set back to the detail data to achieve the desired result. Formatting result set columns with `FORMAT=` and `LABEL=` column modifiers is a SAS enhancement to PROC SQL that is not supported in FedSQL. And finally, it is not uncommon to encounter concatenated libraries in SAS. These libraries have files stored in more than one physical location. The SASHELP library is a good example. The BASE LIBNAME engine can access data sets in concatenated libraries, but the BASE driver for FedSQL (and DS2) can access data only from librefs that point to a single physical location.

1.4 Efficiently Working with DBMS Data

SAS/Access Engine: Supported SAS Functions

Using SASTRACE, you can see when the SAS/ACCESS engine successfully translates a SAS function (YEAR) function to a DBMS function.

Partial SAS Log

```
ORACLE_8: Prepared: on connection 0
select COUNT(*) as ZSQL1, MIN(TXT_1."CUSTOMER_ID") as ZSQL2, COUNT(*) as ZSQL3,
COUNT(TXT_1."TOTAL_RETAIL_PRICE") as ZSQL4, MIN(TXT_1."TOTAL_RETAIL_PRICE") as ZSQL5,
MAX(TXT_1."TOTAL_RETAIL_PRICE") as ZSQL6, SUM(TXT_1."TOTAL_RETAIL_PRICE") as ZSQL7
from orion.ORDER_FACT TXT_1
where EXTRACT(YEAR FROM TXT_1."ORDER_DATE") = 2010
group by TXT_1."CUSTOMER_ID"
```

35

Copyright © SAS Institute Inc. All rights reserved.



SAS/Access Engine: Non-Supported Functions

When non-supported functions are used, they cannot be executed in-database. The data is then be pulled to the Compute Server for processing.

```
select  Country
        , Count(*) as Nobs
        , MIN(TOTAL_RETAIL_PRICE) as Min
        , AVG(TOTAL_RETAIL_PRICE) as Mean
        , MAX(TOTAL_RETAIL_PRICE) as Max
from db.BIG_ORDER_FACT o
inner join
    db.BIG_CUSTOMER_DIM c
on o.Customer_ID=c.Customer_ID
where weekday(datepart(order_date)) = 7
group by COUNTRY
order by COUNTRY
;
```

36

Copyright © SAS Institute Inc. All rights reserved.



PROC SQL - Unsupported Functions for DBMS

The WEEKDAY function cannot be translated. All rows retrieved to SAS:

```
SAS_SQL: Unable to convert the query to a DBMS specific SQL statement due to an error.
ACCESS ENGINE: SQL statement was not passed to the DBMS, SAS will do the processing.
...
ORACLE_34: Prepared: on connection 0
SELECT "CUSTOMER_ID", "TOTAL_RETAIL_PRICE", "ORDER_DATE" FROM student.BIG_ORDER_FACT

ORACLE_35: Executed: on connection 0
SELECT statement ORACLE_34

ORACLE_36: Prepared: on connection 0
SELECT "CUSTOMER_ID", "COUNTRY" FROM student.BIG_CUSTOMER_DIM

ORACLE_37: Executed: on connection 0
SELECT statement ORACLE_36
```

- SAS brings back all rows but only necessary columns.
- Subsetting and join happen in SAS

37

Copyright © SAS Institute Inc. All rights reserved.



PROC FedSQL – Unsupported Functions for DBMS

```
options sastrace=',,,ds' sastraceloc=saslog nostsuffix fullstimer;

proc FedSQL;
select  Country
       , Count(*) as Nobs
       , MIN(TOTAL_RETAIL_PRICE) as Min
       , AVG(TOTAL_RETAIL_PRICE) as Mean
       , MAX(TOTAL_RETAIL_PRICE) as Max
from db.BIG_ORDER_FACT o
   inner join
   db.BIG_CUSTOMER_DIM   c
on o.Customer_ID=c.Customer_ID
where weekday(datepart(order_date)) = 7
group by COUNTRY
order by COUNTRY
;
quit;

options sastrace=off stsufffix nofullstimer;
```

38

Copyright © SAS Institute Inc. All rights reserved.



FedSQL Query Log

```

669 options sastrace=',,,ds' sastraceloc=saslog nostsuffix fullstimer;
670
671 proc FedSQL;
672 select Customer_ID
673        , Count(*) as Nobs
674        , MIN(TOTAL_RETAIL_PRICE) as Min
675        , AVG(TOTAL_RETAIL_PRICE) as Mean
676        , MAX(TOTAL_RETAIL_PRICE) as Max
677 from orion.ORDER_FACT
678 where weekday(datepart(order_date))= '7'
679 group by CUSTOMER_ID
680 order by customer_id
681 ;
682 quit;

```

Despite engaging SASTRACE, there is no indication of DBMS push-down in the SAS log.

```

NOTE: PROCEDURE FedSQL used (Total process time):
      real time           0.14 seconds
      user cpu time       0.00 seconds
      system cpu time     0.03 seconds
      memory              7160.18k
      OS Memory           29256.00k

```

39

Copyright © SAS Institute Inc. All rights reserved.



_METHOD option

PROC FedSQL supports the _METHOD and NOEXEC invocation options.

```

proc FedSQL _method noexec;
<FedSQL code here>;
quit;

```

Use NOEXEC to evaluate your code without executing the query.
For more information about using _METHOD, read :

- [PROC FedSQL Options](#)
- [The SQL Optimizer Project: Method and Tree in SAS®9.1](#)

40

Copyright © SAS Institute Inc. All rights reserved.



Using _METHOD with FedSQL

```
proc FedSQL _method noexec;
select Customer_ID
      , Count(*) as Nobs
      , MIN(TOTAL_RETAIL_PRICE) as Min
      , AVG(TOTAL_RETAIL_PRICE) as Mean
      , MAX(TOTAL_RETAIL_PRICE) as Max
from orion.ORDER_FACT
where weekday(datepart(order_date))= '7'
group by CUSTOMER_ID
order by customer_id
;
quit;
```

Partial SAS Log

Aggregation, sort & join
performed by SAS

```
Methods:
Number of Sorts Performed is : 1
Number of Joins Performed is : 1
Number of Hash Joins Performed is : 1
Limit
HashJoin (INNER)
  SubqueryScan
    Agg
      Sort
        SeqScan with _pushed_ qual from DB.STUDENT.WIDE
        SeqScan from SAS.SAS.WIDE
```

SAS

Undocumented IPTRACE option

PROC FedSQL supports the undocumented IPTRACE invocation option.

```
proc FedSQL iptrace noexec;
<FedSQL code here>;
quit;
```

Use NOEXEC and IPTRACE to see how the code will
interact with the DBMS without executing the query.

Undocumented IPTRACE option

```
proc FedSQL iptrace noexec;
select  Country
      , Count(*) as Nobs
      , MIN(TOTAL_RETAIL_PRICE) as Min
      , AVG(TOTAL_RETAIL_PRICE) as Mean
      , MAX(TOTAL_RETAIL_PRICE) as Max
from db.BIG_ORDER_FACT o
      inner join
      db.BIG_CUSTOMER_DIM c
on o.Customer_ID=c.Customer_ID
where weekday(datepart(order_date)) = 7
group by COUNTRY
order by COUNTRY
;
quit;
```

43

Copyright © SAS Institute Inc. All rights reserved.



PROC FedSQL – Undocumented IPTRACE option

The WEEKDAY function cannot be translated. All rows retrieved to SAS:

```
IPTRACE: Query:
select Country , Count(*) as Nobs , MIN(TOTAL_RETAIL_PRICE) as Min
, AVG(TOTAL_RETAIL_PRICE) as Mean , MAX(TOTAL_RETAIL_PRICE) as
Max from db.BIG_ORDER_FACT o inner join db.BIG_CUSTOMER_DIM c
on o.Customer_ID=c.Customer_ID where weekday(datepart(order_date)) =
7 group by COUNTRY order by COUNTRY
IPTRACE: FULL pushdown to ORACLE FAILURE!
ERROR: [42S22] Column not found ORA-00904: "WEEKDAY": invalid identifier
...
IPTRACE: Minimal pushdown to ORACLE SUCCESS!
IPTRACE: Retextualized child query:
select "O"."CUSTOMER_ID", "O"."ORDER_DATE", "O"."TOTAL_RETAIL_PRICE"
from "STUDENT"."BIG_ORDER_FACT" "O"
IPTRACE: Minimal pushdown to ORACLE SUCCESS!
IPTRACE: Retextualized child query:
select "C"."CUSTOMER_ID", "C"."COUNTRY" from "STUDENT"."BIG_CUSTOMER_DIM" "C"
IPTRACE: END
```

- SAS will bring back all rows but only necessary columns.
- Subsetting and join will happen in SAS



Explicit Pass-Through with PROC SQL

General form of explicit pass-through using PROC SQL:

```
PROC SQL;
  CONNECT TO DBMS (required connect information);
  SELECT column-1, column-2,...column-n
    FROM CONNECTION TO DBMS (DBMS-specific SQL query);
  EXECUTE (DBMS-specific DDL SQL statement) BY DBMS;
  DISCONNECT FROM DBMS;
QUIT;
```

Because PROC SQL uses CONNECT and DISCONNECT statements to execute explicit pass-through SQL statements on the DBMS, no LIBNAME statement is necessary.

45

Copyright © SAS Institute Inc. All rights reserved.



Explicit Pass-Through with PROC SQL

Rewrite the process using explicit SQL.

```
proc sql;
connect to oracle (user=student pw=Metadata0);
select * from
  connection to oracle
  (select /*+full parallel(8)*/
    Country
    , Count(*) as Nobs
    , MIN(TOTAL RETAIL PRICE) as Min
    , AVG(TOTAL RETAIL PRICE) as Mean
    , MAX(TOTAL RETAIL PRICE) as Max
  from student.BIG_ORDER_FACT o
    inner join
      student.BIG_CUSTOMER_DIM c
    on o.Customer ID=c.Customer ID
  where to_char(order_date,'d') = '7'
  group by COUNTRY
  order by COUNTRY)
;
disconnect from oracle;
quit;
```

Connect to DBMS

PROC SQL code
processes result set
from the connection

DBMS-specific SQL
inside parentheses
executes in DBMS.

Disconnect from DBMS

46

Copyright © SAS Institute Inc. All rights reserved.



Explicit Pass-Through with PROC FedSQL

General form of explicit pass-through using PROC FedSQL on the Compute Server:

```
LIBNAME libref dbms_engine <connection options>;
PROC FedSQL;
    SELECT column-1, column-2,...column-n
    FROM CONNECTION TO libref (DBMS-specific SQL query);
    EXECUTE (DBMS-specific DDL SQL statement) BY libref;
QUIT;
```

Because PROC FedSQL uses metadata from a libref to make a connection to the DBMS, a LIBNAME is required, but not CONNECT and DISCONNECT statements.

47

Copyright © SAS Institute Inc. All rights reserved.



Explicit Pass-Through with PROC FedSQL

On the Compute Server

```
libname db oracle path="&OracleServer"
        user=student pw=Metadata0 schema=student;
proc FedSQL _method;
select * from
    connection to db
    (select /**full parallel(8)*/
        Country
        , Count(*) as Nobs
        , MIN(TOTAL_RETAIL_PRICE) as Min
        , AVG(TOTAL_RETAIL_PRICE) as Mean
        , MAX(TOTAL_RETAIL_PRICE) as Max
    from student.BIG_ORDER_FACT o
    inner join
        student.BIG_CUSTOMER_DIM c
    on o.Customer_ID=c.Customer_ID
    where to_char(order_date,'d') = '7'
    group by COUNTRY
    order by COUNTRY)
;
quit;
```

PROC FedSQL code processes result set from libref connection

DBMS-specific SQL inside parentheses executes in DBMS.

48

Copyright © SAS Institute Inc. All rights reserved.



Using _METHOD with Explicit FedSQL Pass-Through

The _METHOD output in the SAS log indicates all computations were performed in the DBMS, and the process executed much more quickly.

Partial SAS Log

```
Methods:
      SeqScan from ORION_DB.{Push Down}.Child 1

NOTE: PROCEDURE FedSQL used (Total process time):
      real time          0.11 seconds
      cpu time           0.06 seconds
```

49

Copyright © SAS Institute Inc. All rights reserved.

sqm03d14



Explicit Pass-Through with PROC FedSQL in CAS

General form of explicit pass-through using PROC FedSQL in CAS:

```
PROC FedSQL sessref=cas-session-name;
SELECT column-1, column-2,...column-n
FROM CONNECTION TO caslib
(DBMS-specific SQL query);
QUIT;
```

While running in CAS, FedSQL has access only to CAS resources, not to Compute Server resources.

- Metadata from the caslib is used to connect to the DBMS.
- DBMS tables need not be loaded into CAS memory for FedSQL to access them.

50

Copyright © SAS Institute Inc. All rights reserved.



Explicit Pass-Through with PROC FedSQL

Bypass the Compute Server – run in CAS

```
proc FedSQL sessref=cs;
select * from
  connection to db2CAS
  (select /*+full parallel(8)*/
    Country
    , Count(*) as Nobs
    , MIN(TOTAL_RETAIL_PRICE) as Min
    , AVG(TOTAL_RETAIL_PRICE) as Mean
    , MAX(TOTAL_RETAIL_PRICE) as Max
  from student.BIG_ORDER_FACT o
    inner join
      student.BIG_CUSTOMER_DIM c
  on o.Customer_ID=c.Customer_ID
  where to_char(order_date,'d') = '7'
  group by COUNTRY
  order by COUNTRY)
;
quit;
```

Connection references
a caslib available in the
"cs" CAS session.

DBMS-specific SQL
inside parentheses
executes in DBMS.

51

Copyright © SAS Institute Inc. All rights reserved.



Advantages

Implicit SQL Pass-Through

Transparent use of DBMS tables in SAS.

Use DBMS tables in DATA step and SAS procedures just like SAS datasets.

No knowledge of DBMS-specific SQL is required.

Explicit SQL Pass-Through

The DBMS can optimize data summarization, ordering and joins.

Functions and features specific to the DBMS can be used.

SAS and DBMS-specific features can be used in the same query.

You can run DBMS stored procedures and macros.

You can pass-through DDL statements such as DROP TABLE and GRANT.

52

Copyright © SAS Institute Inc. All rights reserved.



Disadvantages

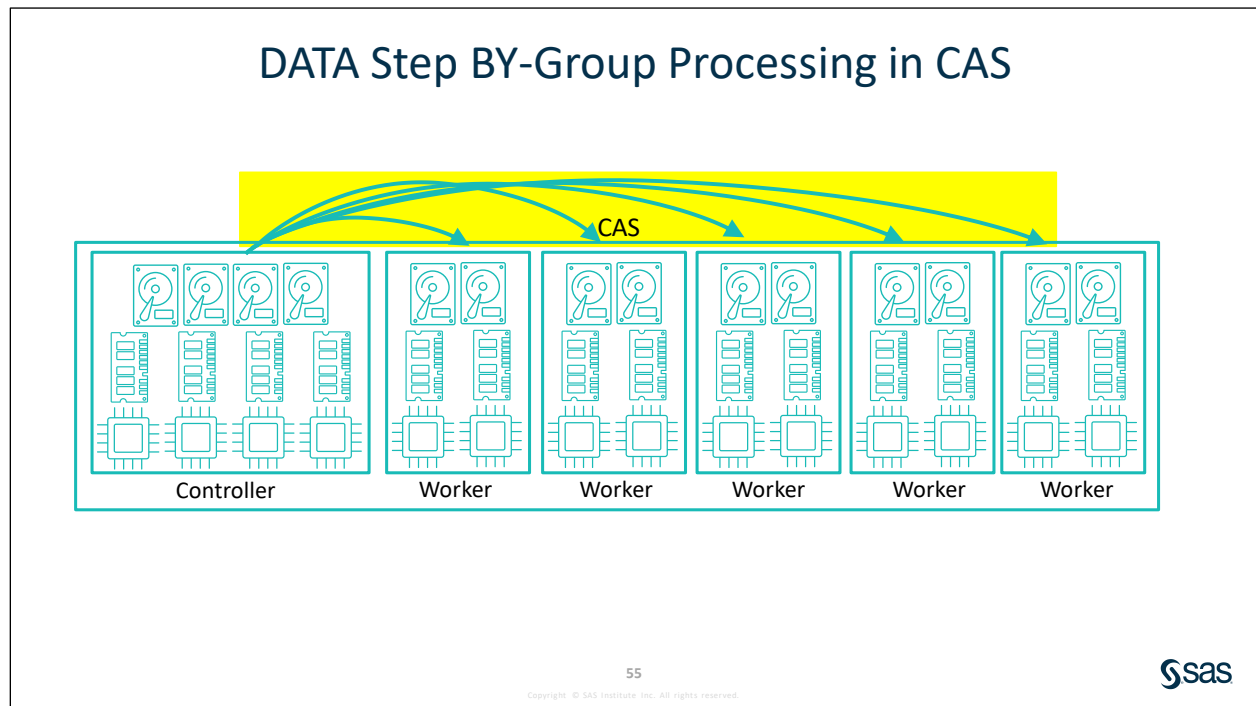
Implicit SQL Pass-Through

Process may inadvertently pull large data volumes to the Compute Server.
Cannot take advantage of DBMS-specific features.

Explicit SQL Pass-Through

Requires DBMS-specific SQL knowledge.

1.5 Joins and Merges



When you process a DATA step in CAS, all rows from a BY group must be located on the same worker node to ensure proper processing. When the data is re-distributed to the nodes, it is distributed by the **formatted** value of the FIRST BY variable. There are two effects of this behavior you should be aware of:

1. You can't use the keyword DESCENDING for the first variable in the BY statement.
2. If the BY variables in the CAS tables being merged have different formats, it is possible that distributed rows would not have the same ID in both tables. CAS will generate an error and stop processing.



Demo: Managing CAS Data with PROC CASUTIL

Program file: **Joins and Merges.sas**

```

/* Joins and merges using DBMS and CAS data */
/* Prepare some data */
data sas.carMPG (keep=ID MPG: )
    db.carMPG (keep=ID MPG: )
    ;
    ID+1;
    set sashelp.cars;
run;

data sas.cars (drop=MPG: )
    ;
    ID+1;
    set sashelp.cars;
    format ID z3.;
run;

proc sort data=sas.cars;
    by Make Model;
run;

data casuser.cars;
    set sas.cars;
run;

/* Using SAS data sets */
data work.merged;
    merge sas.cars
          sas.carMPG;
    by ID;
run;
/* Need to sort! BY variables are not properly sorted on data set
SAS.CARS. */
proc sort data=sas.cars out=work.cars;
    by id;
run;

data work.merged;
    merge work.cars
          sas.carMPG;
    by ID;
run;

proc print data=work.merged(obs=5) ;
run;

```

```

/* NO need to sort CAS or DBMS tables */
data work.merged;
    merge casuser.cars
          db.carMPG;
    by ID;
run;

proc print data=work.merged(obs=5);
run;

/* This DATA step runs on the Compute Server despite sessref=cs*/
data work.merged / sessref=cs;
    merge casuser.cars
          db.carMPG;
    by ID;
run;

/* To run DATA step in CAS, CAS engine librefs must be used for all
data sets */
/* Load Oracle tables to db2cas caslib memory */
proc casutil;
    load casdata="CARMPG"   incaslib="db2cas"
        casout="carMPG"   outcaslib="db2cas" replace;
run;

/* Write output to casuser.merged*/
data casuser.merged /sessref=cs;
    merge casuser.cars
          db2cas.carMPG;
    by ID;
run;

/* Drat! The ID is formatted differently in the two tables */
/* CAS distributes data to workers based on the FORMATTED values of
the BY variables.*/
ods select variables;
proc contents data=casuser.cars (keep=ID);
run;
ods select variables;
proc contents data=db2cas.carMPG (keep=ID);
run;

/* The in-memory table is not the original - just a copy. I can fix
the format */
proc casutil;
    altertable casdata="CARMPG"   incaslib="db2cas"
        COLUMNS={ {NAME="ID", FORMAT="z3."} };
run;

```

```

/* Now my merge works fine */
data casuser.merged / sessref=cs;
    merge casuser.cars
          db2cas.carMPG;
    by ID;
run;

proc print data=casuser.merged(obs=5);
run;

/* What about a FedSQL join? */
/* Let's restore the format to the way it was */
proc casutil;
    altertable casdata="CARMPG" incaslib="db2cas"
        COLUMNS={{NAME="ID",FORMAT=""}};
run;

ods select variables;
proc contents data=casuser.cars (keep=ID);
run;
ods select variables;
proc contents data=db2cas.carMPG (keep=ID);
run;

proc fedSQL sessref=cs;
create table casuser.joined as
select c.*, MPG_City, MPG_Highway
    from casuser.cars as c
    inner join
        db2cas.carmpg as m
    on c.ID=m.ID
;
quit;

/* Oooh - that was much easier! */
proc fedSQL sessref=cs;
title "Joined";
select * from casuser.joined order by id limit 5;
title "Merged";
select * from casuser.merged order by id limit 5;
run;

/* It even works in the CAS action */
proc delete data=casuser.joined;
run;

```

```

proc cas;
source myQ;
create table casuser.joined as
select c.*, MPG_City, MPG_Highway
  from casuser.cars as c
  inner join
    db2cas.carmpg as m
    on c.ID=m.ID
;
endsource;
run;
fedSQL.execDirect /
  query=myQ
;
run;
quit;

proc print data=casuser.joined(obs=5);
run;

/* FedSQL joins of DBMS tables */
/* Compute Server - implicit pass-through */
proc fedSQL;
select
  Country
  , Count(*) as Nobs
  , MIN(TOTAL_RETAIL_PRICE) as Min
  , AVG(TOTAL_RETAIL_PRICE) as Mean
  , MAX(TOTAL_RETAIL_PRICE) as Max
from db.BIG_ORDER_FACT o
  inner join
    db.BIG_CUSTOMER_DIM c
  on o.Customer_ID=c.Customer_ID
where weekday(datepart(order_date)) = '7'
group by COUNTRY
order by COUNTRY
;
quit;

```



```
/* Compute Server - explicit pass-through */
proc fedSQL;
select *
  from connection to db
  (
    select /*+full parallel(8)*/
      Country
      , Count(*) as Nobs
      , MIN(TOTAL_RETAIL_PRICE) as Min
      , AVG(TOTAL_RETAIL_PRICE) as Mean
      , MAX(TOTAL_RETAIL_PRICE) as Max
    from student.BIG_ORDER_FACT o
      inner join
        student.BIG_CUSTOMER_DIM c
    on o.Customer_ID=c.Customer_ID
    where to_char(order_date,'d') = '7'
    group by COUNTRY
    order by COUNTRY
  )
;
quit;

options sastrace=",,,d" sastraceloc=saslog nostsuffix;
proc sql;
select Country
      , Count(*) as Nobs
      , MIN(TOTAL_RETAIL_PRICE) as Min
      , AVG(TOTAL_RETAIL_PRICE) as Mean
      , MAX(TOTAL_RETAIL_PRICE) as Max
from db.BIG_ORDER_FACT o
  inner join
    db.BIG_CUSTOMER_DIM c
on o.Customer_ID=c.Customer_ID
where weekday(datepart(order_date)) = 7
group by COUNTRY
order by COUNTRY
;
quit;
options sastrace=off stsuffix nofullstimer;
```

```
proc FedSQL _method noexec;
select  Country
        , Count(*) as Nobs
        , MIN(TOTAL_RETAIL_PRICE) as Min
        , AVG(TOTAL_RETAIL_PRICE) as Mean
        , MAX(TOTAL_RETAIL_PRICE) as Max
from db.BIG_ORDER_FACT o
      inner join
      db.BIG_CUSTOMER_DIM c
on o.Customer_ID=c.Customer_ID
where weekday(datepart(order_date)) = 7
group by COUNTRY
order by COUNTRY
;
quit;

proc FedSQL IPTRACE noexec;
select  Country
        , Count(*) as Nobs
        , MIN(TOTAL_RETAIL_PRICE) as Min
        , AVG(TOTAL_RETAIL_PRICE) as Mean
        , MAX(TOTAL_RETAIL_PRICE) as Max
from db.BIG_ORDER_FACT o
      inner join
      db.BIG_CUSTOMER_DIM c
on o.Customer_ID=c.Customer_ID
where weekday(datepart(order_date)) = 7
group by COUNTRY
order by COUNTRY
;
quit;
```

```

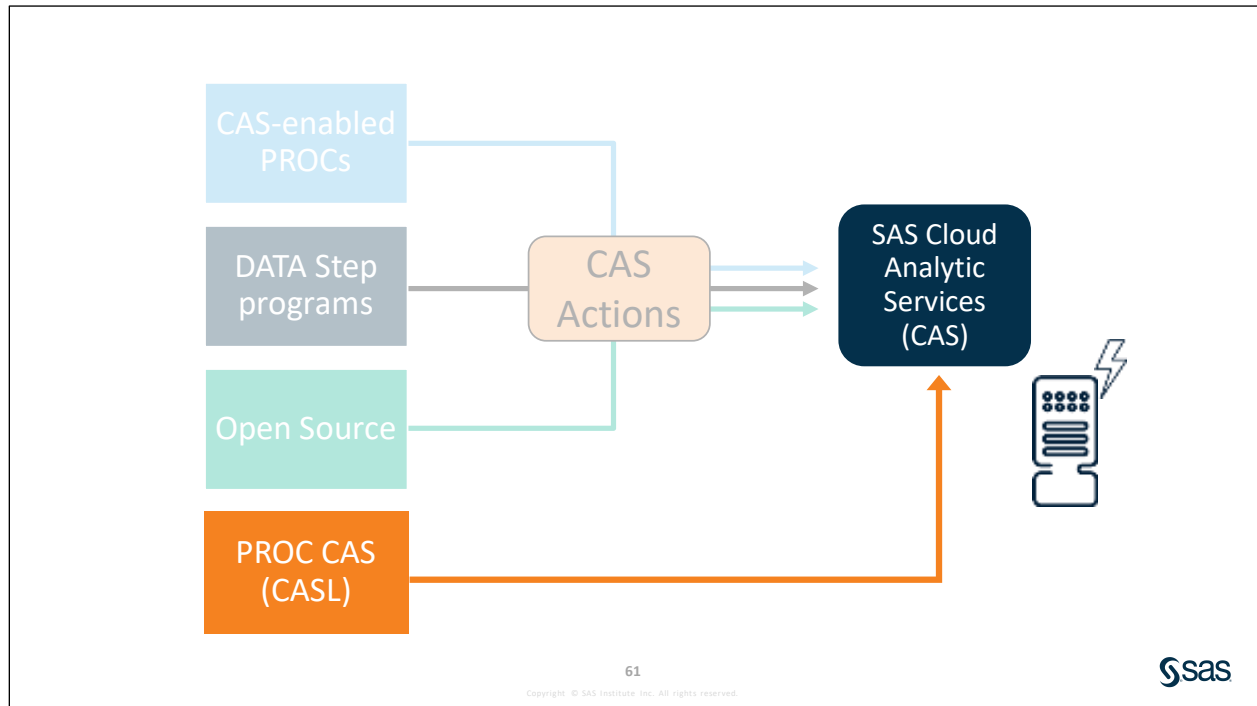
/* FedSQL does not require DBMS tables to be pre-loaded */
/* CAS - implicit pass-through */
proc fedSQL sessref=cs;
select
    Country
    , Count(*) as Nobs
    , MIN(TOTAL_RETAIL_PRICE) as Min
    , AVG(TOTAL_RETAIL_PRICE) as Mean
    , MAX(TOTAL_RETAIL_PRICE) as Max
from db2cas.BIG_ORDER_FACT o
    inner join
    db2cas.BIG_CUSTOMER_DIM c
on o.Customer_ID=c.Customer_ID
where weekday(datepart(order_date)) = '7'
group by COUNTRY
order by COUNTRY
;
quit;

/* CAS - explicit pass-through */
proc fedSQL sessref=cs;
select *
from connection to db2cas
(
    select /*+full parallel(8)*/
        Country
        , Count(*) as Nobs
        , MIN(TOTAL_RETAIL_PRICE) as Min
        , AVG(TOTAL_RETAIL_PRICE) as Mean
        , MAX(TOTAL_RETAIL_PRICE) as Max
    from student.BIG_ORDER_FACT o
        inner join
        student.BIG_CUSTOMER_DIM c
    on o.Customer_ID=c.Customer_ID
    where to_char(order_date,'d') = '7'
    group by COUNTRY
    order by COUNTRY
)
;
quit;

```

End of Demonstration

1.6 PROC CAS



Up until now, we've primarily used CAS-enabled PROCs to get our processing done in CAS. Sometimes, our code resulted in large amounts of data being brought back to the Compute Server for processing. How can I ensure that my code will **always** run in CAS? You can use PROC CAS to write CASL programs. The code you submit using PROC CAS contains CAS actions that can run only in CAS, so execution by the CAS server is guaranteed.



Demo: Executing CASL Code with PROC CAS

Program file: `proc cas.sas`

```
/* Loading data with PROC CASUTIL */
proc casutil;
    load casdata="toyotas.csv" incaslib="casuser"
        casout="Toyotas" outcaslib="casuser" replace;
    contents casdata="Toyotas" incaslib="casuser";
run;
cas cs listhistory 5;

/* Grab the loadTable and columnInfo action code from the log and
paste here */
proc cas;
table.loadTable / path='toyotas.csv', caslib='CASUSER(student)',
casOut={name='Toyotas',
        caslib='CASUSER(student)', replace=true};
table.columnInfo / table={name='Toyotas',
caslib='CASUSER(student)', singlePass=false};
quit;

cas cs listhistory 2;

/* Simplifying coding and program maintenance with CAS variables */
/* Using CAS variables for values that need to be repeated in code */
proc cas;
/* myTable={name='toyotas',caslib='CASUSER'}; */
myTable={name='cars',caslib='CASUSER'};
stats={name=myTable.name||"Summary",caslib="casuser"};

table.columnInfo result=rci/
    table=myTable
;
describe rci;
columns=rci.columnInfo.where(Column like "M%" and
Type='double') [,"Column"];
simple.summary /
    table=myTable
    ,inputs=columns
    ,casout=stats||{replace=true}
;
table.fetch /
    table=stats
;
quit;
```

```

/* Using CAS result tables for data-driven programming */
proc cas;
  table.tableInfo result=rti /
    caslib="casuser"
  ;
  /* describe rti; */

  tables=rti.tableInfo[, "Name"];
  /* describe tables; */

  do thisName over tables;
    thisTable={caslib="casuser", name=thisName};
    table.tableDetails / caslib=thisTable.caslib
    name=thisTable.name;
    table.columnInfo / table=thisTable;
    table.fetch / table=thisTable to=5;
  end;
quit;

/* Execute DATA step code directly in CAS */
proc cas;
  source myDATAstep;
  data casuser.ToyotaSummary;
    set casuser.toyotas;
    by DriveTrain;
    if first.drivetrain then do;
      call missing (city,hwy, count);
    end;
    city+MPG_City;
    hwy+MPG_Highway;
    count+1;
    if last.drivetrain then do;
      AvgMPG=round(sum(city,hwy)/count,.1);
      output;
    end;
    keep DriveTrain Count AvgMpg;
  run;
endsource;

dataStep.runCode /
  code=myDATAstep
;

table.fetch /
  table={caslib="casuser", name="ToyotaSummary"}
;
quit;

```

```

proc cas;
/* Get rid of temporary tables using data-driven techniques */
table.tableInfo result=rti /
    caslib="casuser"
;
tables=rti.tableInfo[, "Name"];
do thisName over tables;
    thisTable={caslib="casuser", name=thisName};
    table.dropTable / caslib=thisTable.caslib name=thisTable.name;
end;

/* Get rid of data files isn a caslib's source */
table.fileInfo result=rfi /
    caslib="casuser"
;
files=rfi.fileInfo[, "Name"];
do thisFile over files;
    thisFile={caslib="casuser", name=thisFile};
    table.deletesource / caslib=thisFile.caslib
source=thisFile.name;
end;
quit;

```

End of Demonstration



Demo: Dealing with CASDATA LIMIT Issues

Program file: **DATALIMIT problems.sas**

```

/*****
Setup:
Create a table with 10 million rows
*****/
data casuser.products;
  call streaminit(99);
  do i=1 to 10000000;
    x=rand('normal');
    if x<.10 then Product="A";
    else if x<.30 then Product="B";
    else if x<.60 then Product="C";
    else Product="D";
    Quantity=round(rand('uniform',1,100));
    output;
  end;
  drop x i;
run;

/*****
Task: Graph a very large data set
- Create a vertical bar chart of the 10 million row CAS table
- must override the SGPlot data size limit
*****/
ods graphics / maxobs=10000000;
proc sgplot data=casuser.products;
  vbar Product /
    response=Quantity
    stat=sum categoryorder=respdesc;
  format Quantity comma16.;
run;
ods graphics / reset;

/*****
Problem: CASDATA LIMIT prevents downloading enormous data
*****/

/*****
Solution (not recommended): Override the CASDATA LIMIT
Allows transfer of more data from CAS to Compute Server
This works, but runs way too long (15 seconds)
*****/
/* Find out how big the data is */
proc casutil;
  contents casdata="products" incaslib="casuser";

```



```

quit;

ods graphics / maxobs=10000000;
/* USE the DATALIMIT= data set option to override the system limit
*/
proc sgplot data=casuser.products(datalimit=160000000);
    vbar Product /
        response=Quantity
        stat=sum categoryorder=respdesc;
    format Quantity comma16.;
run;
ods graphics / reset;

/*****
Better solution:
a. Summarize the data in CAS and create a CAS table
b. Graph the summarized CAS table
*****/

proc cas;
* Summarize the CAS table with an action and save the results as a
CAS table *;
simple.summary /
    table={name="products"
            ,caslib="casuser"
            ,groupBy="Product"
            }
    ,input="Quantity"
    ,subSet={"SUM"}
    ,casOut={name="products_sum"
             ,caslib="casuser"
             ,replace=TRUE
             }
;
quit;

* b *;
proc sgplot data=casuser.products_sum;
    vbar Product /
        response=_sum_
        categoryorder=respdesc;
    format _sum_ comma16.;
    label _sum_="Total Quantity";
quit;

/*****
Cleanup: Drop tabs no longer needed
*****/
proc casutil;

```

```
droptable casdata="products" incaslib="casuser" quiet;  
droptable casdata="products_sum" incaslib="casuser" quiet;  
quit;
```

End of Demonstration

1.7 Learning More

Never stop learning! Here are some solid resources to continue your learning experience:

- **SAS Global Forum Papers:**
 - [Anything You Can Do I Can Do Better: PROC FEDSQL VS PROC SQL](#)
 - [Best Practices for Converting SAS® Code to Leverage SAS® Cloud Analytic Services](#)
 - [Parallel Programming with the DATA Step: Next Steps](#)
 - [Working with Big Data in SAS®](#)
- **Docs: [FedSQL Programming](#):**
 - [PROC FedSQL - SQL Enhancements](#)
 - [PROC FedSQL – SQL Limitations](#)
 - [Resolving PROC FedSQL Errors in the Compute Server](#)
- **Docs: [FedSQL programming for CAS](#):**
 - [Resolving FedSQL Errors in CAS](#)

If you have questions specific to your SAS Viya installation, please address them to your internal SAS administrator who is familiar with your system's topology and data structures.

For general questions about SAS programming:

- [SAS Communities](#) forums are a wonderful resource.
- The [SAS Users](#) YouTube Channel is great for targeted videos on specific topics.



Email: Mark.Jordan@sas.com

Twitter: [@SASJedi](https://twitter.com/SASJedi)

Blog: <http://go.sas.com/jedi>

Author Page: <http://support.sas.com/jordan>

