Analyzing Database Tables in SAS® Viya® Using SQL

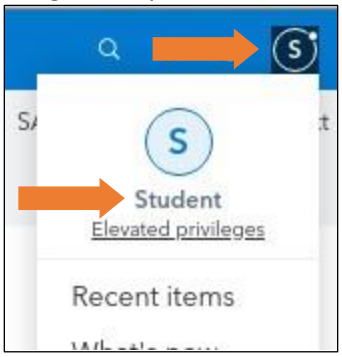
Allison Saito
Technical Training Consultant at SAS

Hands On Workshop Instructions

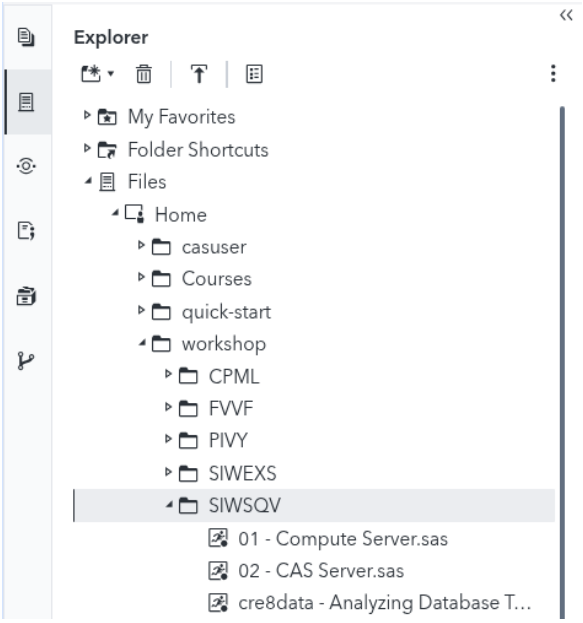
Setup : Sign In and Create Data

1. Open Google Chrome and select the **SAS Studio** bookmark.
2. If necessary, sign in to SAS Viya using the username **student** and the password **Metadata0**.

Note: If you were already signed in to SAS Viya, make sure that you are signed in to the **Student** account. You can check the account by going to the top right of SAS Studio and clicking the circle icon. Then view the account name. If the account isn't **Student,** log out and log back in using the required information.



3. When prompted to opt in to all of your assumable groups, select **No**.
4. In SAS Studio, navigate to **Files** > **Home** > **workshop** > **SIWSQV**.



5. From the SIWSQV folder, open **cre8data – Analyzing Database Tables.sas**. Run the the entire program. The specifics of this program are outside the scope of this workshop. We only need to confirm it ran as intended. View the log and confirm no error messages. View the results, and confirm the following report appears.

Results from table.fileInfo

FileInfo Data Source Entities

Library	Schema	Type	Name
ORAC_CAS	STUDENT	TABLE	CUSTOMERS
ORAC_CAS	STUDENT	TABLE	LOANS_RAW

SAS Compute Server Database Processing

1. In the **SIWSQV** folder, open the **01 – Compute Server.sas** program. The LIBNAME statement connects the Oracle database to the SAS compute server using SAS/ACCESS Interface to Oracle. Run the statement.

```
libname or_db oracle path="//server.demo.sas.com:1521/ORCL"
                     authDomain="OracleAuth"
                     schema="STUDENT";
```

2. The OPTIONS statement turns on options to enable us to see what SQL was sent to the Oracle database for processing. Run the statement.

```
options sasTrace=',,,d' sasTraceLoc=sasLog noStSuffix;
```

3. In the IMPLICIT PASS-THROUGH ON THE SAS COMPUTE SERVER section, we will use SAS implicit pass-through to process the Oracle database table. With implicit pass-through, SAS attempts to convert SAS PROC SQL syntax into database SQL wherever possible. If it can't convert the SQL to database SQL, it brings the data to the SAS compute server for processing.

    a. Run the CONTENTS procedure to view all available tables in the Oracle database.

```
proc contents data=or_db._all_ noDs;
run;
```

View the results. Notice that there are two database tables, the **CUSTOMERS** table and the **LOANS_RAW** table.

#	Name	Member Type	DBMS Member Type
1	CUSTOMERS	DATA	TABLE
2	LOANS_RAW	DATA	TABLE

View the log. After the text of the PROC CONTENTS, notice the lines beginning *ORACLE_X: Prepared*. *ORACLE_X* tells us the step's library is connected to Oracle, and *X* will vary depending on how many queries have been submitted in this session. *Prepared* tells us that the next row begins a proposed query to send to Oracle.

After the prepared query, there is another pair of lines. *Oracle_X: Executed* tells us that a query executed on Oracle. The next line tells us which query. In the example log, the *ORACLE_1* query executed on Oracle.

The conversion and execution of non-Oracle code into an Oracle SQL query is called implicit pass-through.

```
ORACLE_1: Prepared: on connection 1
SELECT OBJECT_NAME ,OBJECT_TYPE  FROM  ALL_OBJECTS  OBJ WHERE
   (OBJ.OWNER ='STUDENT') AND (OBJ.OBJECT_TYPE  IN ('TABLE',
   'VIEW'))


ORACLE_2: Executed: on connection 1
SELECT statement  ORACLE_1
```

b. The first PROC SQL query previews 10 rows from the **LOANS_RAW** database table. Run the SAS SQL procedure to run the query with the OBS= SAS data set option.

```
proc sql;
select *
   from or_db.loans_raw(obs=10);
quit;
```

View the log. Notice that implicit pass-through converted the SAS SQL query to an Oracle SQL query.

```
ORACLE_3: Prepared: on connection 0
SELECT * FROM STUDENT.LOANS_RAW FETCH FIRST         10 ROWS ONLY

ORACLE_4: Executed: on connection 0
SELECT statement  ORACLE_3
```

c. Next, we will count the total number of rows and total loan amount by **Category** in the **LOANS_RAW** database table. The first query will use implicit pass-through. The second query will disable implicit pass-through using the NOIPASSTHRU option. Both queries will produce identical results. Run both queries.

```
proc sql;
select Category,
       count(*) as TotalLoansByCategory format=comma16.,
       sum(Amount) as TotalAmount format=dollar20.2
    from or_db.loans_raw
    group by Category
    order by Category;
quit;

proc sql noIPassThru;
 select Category,
       count(*) as TotalLoansByCategory format=comma16.,
       sum(Amount) as TotalAmount format=dollar20.2
     from or_db.loans_raw
     group by Category
     order by Category;
quit;
```

View the log. Notice that the first query was converted into Oracle SQL via implicit pass-through. In the sample log, it ran in approximately eight seconds.

```
ORACLE_5: Prepared: on connection 0
SELECT * FROM STUDENT.LOANS_RAW

ORACLE_6: Prepared: on connection 0
 select TXT_1."Category", COUNT(*) as TotalLoansByCategory,
    SUM(TXT_1."Amount") as TotalAmount from STUDENT.LOANS_RAW
    TXT_1 group by TXT_1."Category" order by TXT_1."Category"
    asc NULLS FIRST

ORACLE_7: Executed: on connection 0
SELECT statement  ORACLE_6
...
NOTE: PROCEDURE SQL used (Total process time):
      real time           7.63 seconds
```

Category	TotalLoansByCategory	TotalAmount
Car Loan	785,883	$22,968,273,990.01
Consolidation	956,983	$19,611,577,759.59
Credit Card	5,944,360	$29,838,760,839.09
Education	448,928	$16,710,940,537.78
Home Improvement	672,629	$4,156,437,799.09
Major Purchase	112,124	$669,326,403.92
Medical	336,312	$12,097,338,287.34
Mortgage	855,370	$318,341,884,551.08
Moving Expenses	55,744	$333,383,431.31
Personal	111,949	$668,522,275.42
Small Business	783,945	$15,849,337,182.12
Vacation	112,218	$672,689,613.81
Weddings	55,978	$334,518,072.38

The log for the second query shows that even though implicit pass-through was disabled, SAS efficiently brings back only the necessary columns to process the query on the SAS compute server. In the sample log, the second query took approximately 12 seconds to process, longer than the implicit pass-through query for the same results.

```
ORACLE_9: Prepared: on connection 0
SELECT  "Category", "Amount" FROM STUDENT.LOANS_RAW

ORACLE_10: Executed: on connection 0
SELECT statement  ORACLE_9

151  quit;
NOTE: The PROCEDURE SQL printed page 8.
NOTE: PROCEDURE SQL used (Total process time):
      real time           11.95 seconds
```

Category	TotalLoansByCategory	TotalAmount
Car Loan	785,883	$22,968,273,990.01
Consolidation	956,983	$19,611,577,759.59
Credit Card	5,944,360	$29,838,760,839.09
Education	448,928	$16,710,940,537.78
Home Improvement	672,629	$4,156,437,799.09
Major Purchase	112,124	$669,326,403.92
Medical	336,312	$12,097,338,287.34
Mortgage	855,370	$318,341,884,551.08
Moving Expenses	55,744	$333,383,431.31
Personal	111,949	$668,522,275.42
Small Business	783,945	$15,849,337,182.12
Vacation	112,218	$672,689,613.81
Weddings	55,978	$334,518,072.38

d. In the next two queries, we will count the number of canceled loans by **Year** that begin with the string *Bad*. The first query uses the SAS SCAN function to search for the string *Bad*. The second query uses the ANSI standard LIKE operator. Both queries will achieve similar results. Run both queries.

```
proc sql;
 select Year,
        count(*) as TotalCancelled_BAD format=comma16.
    from or_db.loans_raw
    where scan(CancelledReason,1) = 'Bad'
    group by Year
    order by Year desc;
quit;

proc sql;
 select Year,
         count(*) as TotalCancelled_BAD format=comma16.
    from or_db.loans_raw
    where CancelledReason like 'Bad %'
    group by Year
    order by Year desc;
quit;
```

View the log and results. Notice that the first query attempts to use implicit pass-through, but it could not convert the query to Oracle SQL. Instead, it wrote a query to bring only the necessary columns to the SAS compute server for processing. In the sample log, this query is ORACLE_12. In the sample log, it took approximately seven seconds to execute.

```
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_12: Prepared: on connection 0
SELECT  "Year", "CancelledReason" FROM STUDENT.LOANS_RAW

ORACLE_13: Executed: on connection 0
SELECT statement  ORACLE_12

170  quit;
NOTE: The PROCEDURE SQL printed page 9.
NOTE: PROCEDURE SQL used (Total process time):
```

```
      real time            7.12 seconds
```

Year	TotalCancelled_BAD
2024	2,979
2023	2,358
2022	1,889
2021	2,476
2020	2,241
2019	1,738
2018	1,221
2017	981
2016	668
2015	335

View the log for the second query. Notice that the SQL procedure syntax was converted into Oracle SQL through SAS implicit pass-through and the SELECT statement was passed to the Oracle database for processing. Only the smaller, summarized results were returned to SAS. In the sample log, this query ran in approximately three seconds, less than half the time that it took the previous query to run.

```
ORACLE_15: Prepared: on connection 0
 select TXT_1."Year", COUNT(*) as TotalCancelled_BAD from
   STUDENT.LOANS_RAW TXT_1 where TXT_1."CancelledReason" like
   'Bad %' group by TXT_1."Year" order by TXT_1."Year" desc
   NULLS LAST

ORACLE_16: Executed: on connection 0
SELECT statement  ORACLE_15

ACCESS ENGINE:  SQL statement was passed to the DBMS for
   fetching data.
183  quit;
NOTE: The PROCEDURE SQL printed page 10.
NOTE: PROCEDURE SQL used (Total process time):
      real time           2.66 seconds
```

Year	TotalCancelled_BAD
2024	2,979
2023	2,358
2022	1,889
2021	2,476
2020	2,241
2019	1,738
2018	1,221
2017	981
2016	668
2015	335

e. The last query in this section uses the SAS YEAR and DATEPART functions to summarize by year of the **LastPurchase** datetime column. Run the query.

```
proc sql;
select year(datepart(LastPurchase)) as
LastPurchaseYear,
       count(*) as Total format=comma16.,
       count(*)/(select count(*)
                     from or_db.loans_raw
                     where Category = 'Credit Card')
          as LastPurchasePct format=percent7.1
    from or_db.loans_raw
    where Category = 'Credit Card'
    group by LastPurchaseYear
    order by Total desc;
quit;
```

View the log. Notice that SAS implicit pass-through was unable to convert the entire query to database SQL. Though it states this is "due to an error", it is not a showstopping issue. It means the query had a feature it was unable to translate, in this case the DATEPART and YEAR functions.

While the entire query is not converted to Oracle SQL, implicit pass-through converted the subquery (ORACLE_19) to Oracle SQL to run in the database. It also converted the main query (ORACLE_20) to select only the necessary columns (**LastPurchase** and **Category**) and rows (*Credit Card*) to bring back to the SAS compute server for processing. In the sample log, this query took approximately 11 seconds to run.

```
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_19: Prepared: on connection 0
 select COUNT(*) from STUDENT.LOANS_RAW TXT_2 where
    TXT_2."Category" = 'Credit Card'

ORACLE_20: Prepared: on connection 0
SELECT  "LastPurchase", "Category" FROM STUDENT.LOANS_RAW
    WHERE  ("Category" = 'Credit Card' )

ORACLE_21: Executed: on connection 0
SELECT statement  ORACLE_20

ORACLE_22: Executed: on connection 0
SELECT statement  ORACLE_19

ACCESS ENGINE:  SQL statement was passed to the DBMS for
    fetching data.
209  quit;
NOTE: The PROCEDURE SQL printed page 11.
NOTE: PROCEDURE SQL used (Total process time):
      real time           11.36 seconds
```

LastPurchaseYear	Total	LastPurchasePct
2024	4,938,374	83.1%
2023	364,640	6.1%
2022	243,836	4.1%
2021	167,045	2.8%
2020	103,677	1.7%
2019	61,795	1.0%
2018	35,944	0.6%
2017	19,227	0.3%
2016	7,889	0.1%
2015	1,933	0.0%

4. In the EXPLICIT PASS-THROUGH ON THE SAS COMPUTE SERVER section, we will use explicit pass-through to write and submit Oracle SQL using the SQL procedure. This enables us to use database features, and it will ensure that the query runs inside the database.
    a. The following query uses Oracle SQL (explicit pass-through) to achieve the same results as the previous query. Run the query.

```
proc sql;
/* Connect to the Oracle database */
connect using or_db;

/* Use SAS formats for the results from the Oracle query */
select LastPurchaseYear,
        Total format=comma16.,
       LastPurchasePct format=percent7.1
   from connection to or_db
    (
     select EXTRACT(YEAR FROM "LastPurchase") as
                 LastPurchaseYear,
           count(*) as Total,
           count(*)/(select count(*)
                       from loans_raw
                       where "Category" = 'Credit Card')
                    as  LastPurchasePct
        from loans_raw
        where "Category" = 'Credit Card'
        group by EXTRACT(YEAR FROM "LastPurchase")
        order by Total desc
    );

/* Disconnect from the Oracle database */
disconnect from or_db;
quit;
```

View the log. Notice that SAS sent the Oracle SQL query directly to the database for processing. In the sample log, this explicit pass-through query ran in approximately six seconds, or about half of the time as the equivalent implicit pass-through query in part 3e.

```
ORACLE_23: Prepared: on connection 2
select EXTRACT( YEAR FROM "LastPurchase") as LastPurchaseYear,
   count(*) as Total, count(*)/(select count(*) from loans_raw
   where "Category" = 'Credit Card') as LastPurchasePct from
   loans_raw where "Category" = 'Credit Card' group by
   EXTRACT(YEAR FROM "LastPurchase") order by Total desc


ORACLE_24: Executed: on connection 2
SELECT statement  ORACLE_23
...
NOTE: The PROCEDURE SQL printed page 12.
```

```
NOTE: PROCEDURE SQL used (Total process time):
      real time           6.44 seconds
```

LASTPURCHASEYEAR	TOTAL	LASTPURCHASEPCT
2024	4,938,374	83.1%
2023	364,640	6.1%
2022	243,836	4.1%
2021	167,045	2.8%
2020	103,677	1.7%
2019	61,795	1.0%
2018	35,944	0.6%
2017	19,227	0.3%
2016	7,889	0.1%
2015	1,933	0.0%

CAS Server Database Processing

1. Open the **02 – CAS Server.sas** program. The CAS statement makes a connection to the CAS server from the compute server. The CASLIB statement creates a CAS server connection to the same Oracle database that we used earlier. Run the CAS and CASLIB statements.

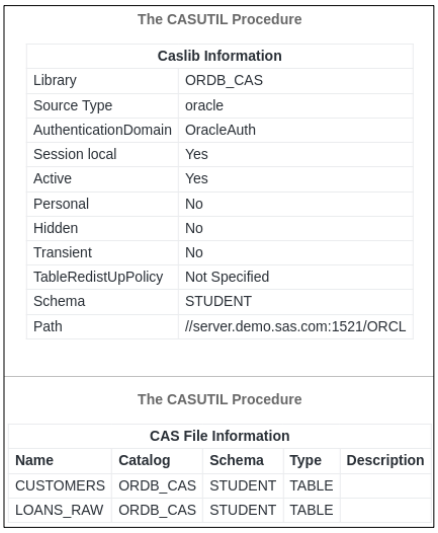
```
cas conn;

caslib ordb_cas dataSource=(srctype="oracle",
                            authDomain="OracleAuth",
                            path="//server.demo.sas.com:1521/ORCL",
                            schema="STUDENT");
```

2. View available data source files and in-memory CAS tables in the **ordb_cas** caslib by running the CASUTIL procedure with the LIST FILES and LIST TABLES statements. Run the step.

```
proc casUtil inCaslib = 'ordb_cas';
   list files;   * View available database tables *;
   list tables;  * View available in-memory CAS tables *;
quit;
```

View the results and log. Notice that we have access to the same Oracle database with the **LOANS_RAW** and **CUSTOMERS** tables. We also see that no tables are loaded into memory on the CAS server.



The CASUTIL Procedure

Caslib Information

Library	ORDB_CAS
Source Type	oracle
AuthenticationDomain	OracleAuth
Session local	Yes
Active	Yes
Personal	No
Hidden	No
Transient	No
TableRedistUpPolicy	Not Specified
Schema	STUDENT
Path	//server.demo.sas.com:1521/ORCL

The CASUTIL Procedure

CAS File Information

Name	Catalog	Schema	Type	Description
CUSTOMERS	ORDB_CAS	STUDENT	TABLE	
LOANS_RAW	ORDB_CAS	STUDENT	TABLE	

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

3. Next, we will use implicit pass-through in CAS using a caslib. This process is similar to the compute server process. The main difference is that we will use the FEDSQL procedure with the SESSREF= option instead of the SQL procedure. For implicit pass-through to be used, we must reference the database table name, not an in-memory CAS table. In this example, **ordb_cas.loans_raw** is not in-memory in CAS. It is in Oracle. Run the FEDSQL procedure.

```
proc fedSql sessref=conn;
select Category,
       count(*)as TotalLoansByCategory,
        sum(Amount) as TotalAmount
     from ordb_cas.loans_raw
     group by Category;
quit;
```

View the log and results. The log shows that the SQL statement was fully offloaded to the underlying data source via full pass-through.

```
NOTE: The SQL statement was fully offloaded to the underlying data source via full pass-
through
```

Category	TOTALLOANSBYCATEGORY	TOTALAMOUNT
Major Purchase	112124	669326404
Weddings	55978	334518072
Moving Expenses	55744	333383431
Mortgage	855370	318341884551
Vacation	112218	672689614
Personal	111949	668522275
Credit Card	5944360	29838760839
Home Improvement	672629	4156437799
Consolidation	956983	19611577760
Small Business	783945	15849337182
Medical	336312	12097338287
Car Loan	785883	22968273990
Education	448928	16710940538

4. In this next example, we will use explicit pass-through with a caslib to subset the database table using Oracle SQL. The syntax for explicit pass-through in FEDSQL is similar to that used in PROC SQL, except CONNECT and DISCONNECT statements are not needed in FEDSQL. FEDSQL uses the connection information that is already stored in the caslib. In this example, the SAS CREATE TABLE statement creates an in-memory CAS table with the results from Oracle. Run the FEDSQL procedure.

```
proc fedSql sessRef=conn;
create table ordb_cas.CCAccounts2022{options replace=true} as
  select *
  from connection to ordb_cas
     (
   /* Send native Oracle query directly to the database for
      processing */
      select *
        from loans_raw
        where EXTRACT(YEAR FROM "LastPurchase") = 2022 and
              "Category" = 'Credit Card'
     );
quit;
```

View the log. It does not mention offloading. We know the query was run in Oracle, because explicit pass-through either runs in the database or errors out. This did not error out. It made (or overwrote) CCACCOUNTS2022 as an in-memory table in the ORDB_CAS caslib.

```
NOTE: Table CCACCOUNTS2022 was created in caslib ORDB_CAS with 243836 rows returned.
```

5. In the Analyze a CAS table section, we will process an in-memory CAS table using a variety of methods. This builds off of part 4. If part 4 has not run and created **ordb_cas.CCAccounts2022**, run it now.

a. First, the CASUTIL procedure uses the LIST TABLES statement to list available CAS tables in the **ordb_cas** caslib. The CONTENTS statement shows the metadata of the **CCAccounts2022** CAS table that we created earlier. Run the CASUTIL procedure.

```
proc casutil;
    list tables incaslib='ordb_cas';
    contents casdata="CCAccounts2022" incaslib="ordb_cas";
quit;
```

b. Once a file is loaded into CAS, it can be processed in-memory in CAS. We can use the FEDSQL procedure with the SESSREF= option to execute SQL queries on a CAS table. Here, we run two simple queries on the CAS table, and the processing is done in CAS. Run the FEDSQL procedure.

```
proc fedsql sessref=conn;
 select *
    from ordb_cas.CCAccounts2022
    limit 10;

select LoanGrade,
        count(*) as TotalLoansByCategory,
        sum(Amount) as TotalAmount
     from ordb_cas.CCAccounts2022
     group by LoanGrade
     order by LoanGrade;
quit;
```

View the log. Since a CAS table and sessRef= were used, the queries either ran in CAS or errored out. These did not error out, so we know they ran in CAS.

c. Once a table is loaded into memory, we can also execute CAS actions to process the data. For example, we can use the simple.summary CAS action to obtain descriptive statistics. Run the CAS procedure.

```
proc cas;
simple.summary /
     table = {name = 'CCAccounts2022',
              caslib = 'ordb_cas'};
quit;
```

View the log and results. In the log, notice that the summary action is processed in under half a second and returns a variety of descriptive statistics. In the results, note the variety of summary statistics generated.

Descriptive Statistics for CCACCOUNTS2022

Column	Minimum	Maximum	N	Sum	Mean	Std Dev	Std Error	Variance	Coeff of Variation	Corrected SS	USS	t Value	Pr > |t|	N Miss	Skewness	Kurtosis
Year	2015.00	2022.00	243836	492536671	2019.95	1.6853	0.003413	2.8403	0.08343	692564	9.949E11	591845	<.0001	0	-0.9322	0.2787
Month	1.0000	12.0000	243836	1569330	6.4360	3.1981	0.006476	10.2277	49.6904	2493868	12594086	993.75	<.0001	0	0.02276	-1.1604
Day	1.0000	27.0000	243836	3414347	14.0026	7.5086	0.01521	56.3786	53.6225	13747065	61556927	920.88	<.0001	0	-0.00019	-1.1908
Amount	-1319.53	37034	243836	299645232	1228.88	2799.49	5.6693	7837138	227.81	1.911E12	2.279E12	216.76	<.0001	0	2.3867	5.6060
InterestRate	8.4600	32.8500	243836	4207076.02	17.2537	2.7891	0.005648	7.7792	16.1653	1896843	74484519	3054.67	<.0001	0	0.4794	0.2169
LoanLength	999999	999999	243836	2.43836E11	999999	0	0	0	0	0	2.438E17	.	.	0	.	.
LastPurchase	1.9566E9	1.9882E9	243836	4.81153E14	1.9733E9	9077249	18383	8.24E13	0.4600	2.009E19	9.495E23	107345	<.0001	0	-0.1178	-1.1806
Cancelled	0	1.0000	243836	2449	0.01004	0.09971	0.000202	0.009943	992.80	2424.40	2449.00	49.74	<.0001	0	9.8274	94.5776
Promotion	0	0	243836	0	0	0	0	0	.	0	0	.	.	0	.	.

6. Sometimes, people try to use PROC SQL to run queries on an in-memory CAS table. While this is not recommended, it is possible for smaller tables.

   To execute PROC SQL on a CAS table, we first need to create a library reference to a caslib with the CAS engine using the LIBNAME statement. Then, the library can be used to point to in-memory tables of the caslib. During the query, the CAS table is temporarily copied to the compute server for processing. This is done in the background. If the copy is successful, no notes are placed in the log. Run the LIBNAME statement and the SQL procedure.

```
libname ordb_cas cas caslib='ordb_cas';

proc sql;
select Category,
       count(*) as TotalLoansByCategory,
       sum(Amount) as TotalAmount
    from ordb_cas.CCAccounts2022
    group by Category;
quit;
```

Category	TotalLoansByCategory	TotalAmount
Credit Card	243836	2.9965E8

   This CAS table was small enough that it was temporarily copied to the compute server without issue.

7. To show what happens if we attempt PROC SQL with a large CAS table, we must first create a sufficiently large in-memory table in CAS. Then, we will then attempt to query it using PROC SQL. Run the CASUTIL and SQL procedures.

```
proc casUtil;
  load inCaslib='ordb_cas' casData="LOANS_RAW"
      outCaslib='ordb_cas' casOut="loans_raw_in_memory" replace;
quit;

proc sql;
```

```
select Category,
       count(*) as TotalLoansByCategory,
       sum(Amount) as TotalAmount
    from ordb_cas.loans_raw_in_memory
    group by Category;
quit;
```

View the log. Notice that we received an error, because **loans_raw_in_memory** is larger than the DATALIMIT option. The table was too large to background copy to compute.

```
ERROR: The maximum allowed bytes (104857600) of data have been fetched from Cloud Analytic
Services. Use the DATALIMIT option to increase the maximum value.
```

8. If you see the previous error, move the query to PROC FEDSQL with SESSREF=. This has been done for us in the next PROC FEDSQL step. Notice the query is identical to the previous PROC SQL query. Run the FEDSQL procedure.

```
proc fedSql sessRef=conn;
select Category,
       count(*) as TotalLoansByCategory,
       sum(Amount) as TotalAmount
    from ordb_cas.loansA_raw_in_memory
    group by Category;
quit;
```

View the log and results. There should be no errors or warnings. Because the query ran with sessRef=conn and on an in-memory CAS table, it ran in CAS.

Category	TOTALLOANSBYCATEGORY	TOTALAMOUNT
Personal	111949	668522275
Credit Card	5944360	29838760839
Medical	336312	12097338287
Small Business	783945	15849337182
Vacation	112218	672689614
Education	448928	16710940538
Mortgage	855370	318341884551
Moving Expenses	55744	333383431
Weddings	55978	334518072
Home Improvement	672629	4156437799
Car Loan	785883	22968273990
Consolidation	956983	19611577760
Major Purchase	112124	669326404