SAS EXPLORE

Analyzing Database Tables in SAS® Viya® Using SQL

Peter Styliadis
Sr Technical Training Consultant at SAS

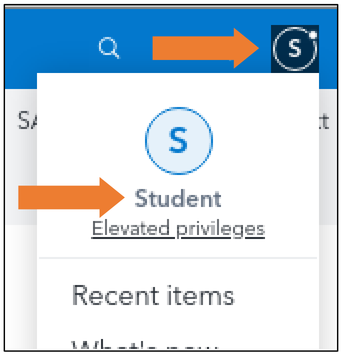
#ExploreSAS

SAS Explore 2023 - Analyzing Database Tables in SAS Viya Using SQL

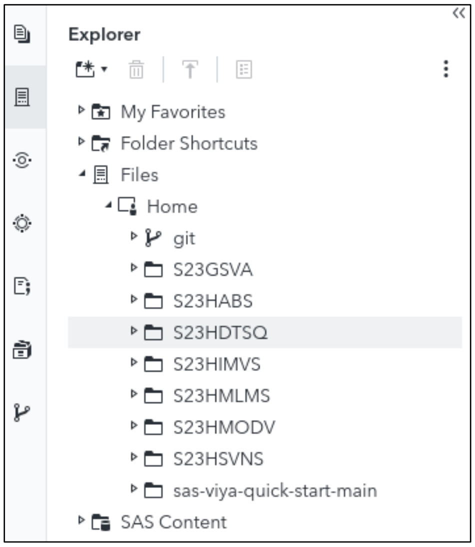
Sign in to SAS Viya and open SAS Studio

1. Open Google Chrome and select the **SAS Studio** bookmark.
2. If necessary, sign in to SAS Viya using the user name **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. In SAS Studio, select **Files** > **Home** > **S23HDTSQ**.



SAS Compute Server Database Processing

1. In the **SQL Workshop** 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.

```
libname or_db oracle path="//server.demo.sas.com:1521/ORCL"
                     user="STUDENT"
                     password="Metadata0"
                     schema="STUDENT";
```

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

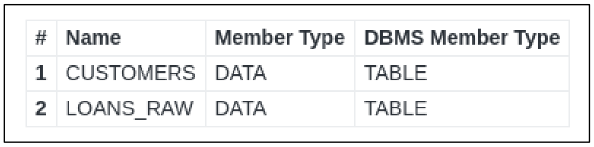
```
options sastrace=',,,d' sastraceloc=saslog nostsuffix
        sql_ip_trace=(note, source);
```

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 native PROC SQL syntax into native database SQL wherever possible. If it can't convert the SQL to native database SQL, it brings the data to the SAS Compute Server for the processing.

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

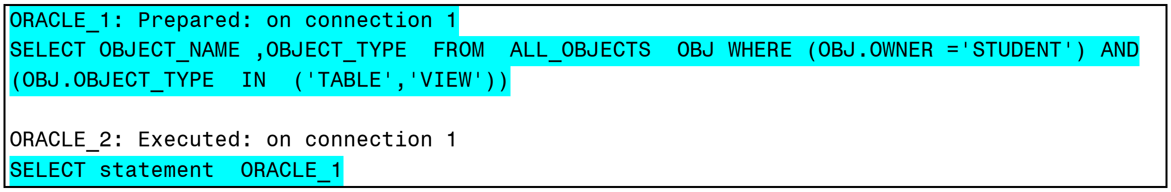
```
proc contents data=or_db._all_ nods;
run;
```

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. Notice that SAS converted the CONTENTS procedure to a native Oracle SQL query using implicit pass-through.

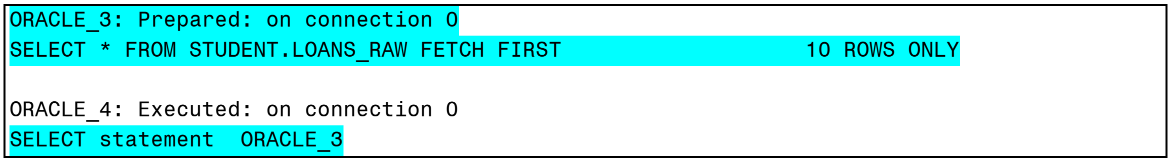


```
ORACLE_1: Prepared: on connection 1
SELECT OBJECT_NAME ,OBJECT_TYPE  FROM  ALL_OBJECTS  OBJ WHERE (OBJ.OWNER ='STUDENT') AND
(OBJ.OBJECT_TYPE  IN  ('TABLE','VIEW'))

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

b. The first PROC SQL query will preview 10 rows from the **LOANS_RAW** database table. We will use 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 a native Oracle SQL query.



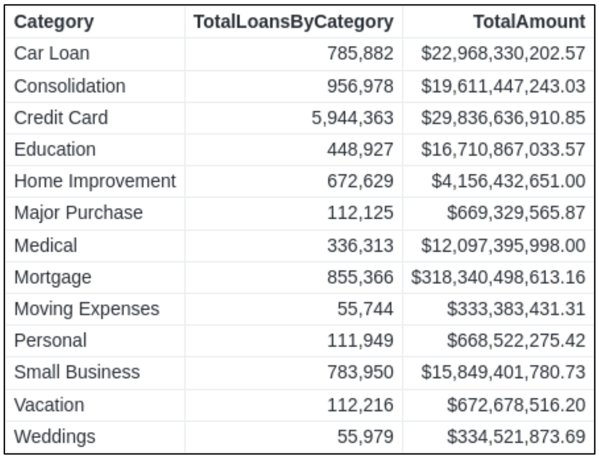
```
ORACLE_3: Prepared: on connection 0
SELECT * FROM STUDENT.LOANS_RAW FETCH FIRST                   10 ROWS ONLY

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 native Oracle SQL via implicit pass-through and ran in approximately three seconds.

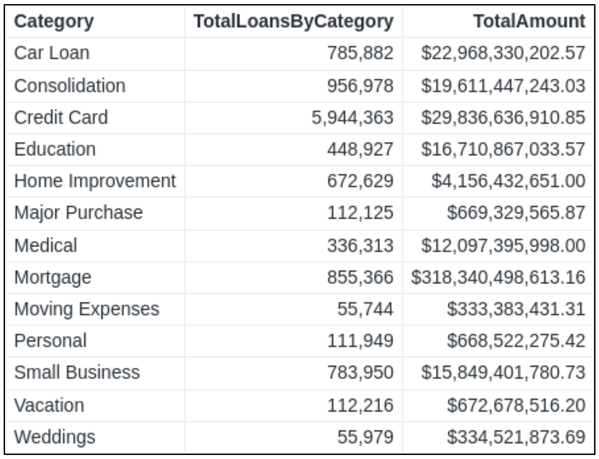
```
ORACLE_18: Prepared: on connection 0
 select TXT_1."Category", COUNT(*) as TotalLoansByCategory, SUM(TXT_1."Amount") as
TotalAmount from STUDENT.LOANS_RAW TXT_1 group by TXT_1."Category" order by
TXT_1."Category" as NULLS FIRST
...
SQL_IP_TRACE: The SELECT statement was passed to the DBMS.
ORACLE_19: Executed: on connection 0
SELECT statement ORACLE_18
```



Category	TotalLoansByCategory	TotalAmount
Car Loan	785,882	$22,968,330,202.57
Consolidation	956,978	$19,611,447,243.03
Credit Card	5,944,363	$29,836,636,910.85
Education	448,927	$16,710,867,033.57
Home Improvement	672,629	$4,156,432,651.00
Major Purchase	112,125	$669,329,565.87
Medical	336,313	$12,097,395,998.00
Mortgage	855,366	$318,340,498,613.16
Moving Expenses	55,744	$333,383,431.31
Personal	111,949	$668,522,275.42
Small Business	783,950	$15,849,401,780.73
Vacation	112,216	$672,678,516.20
Weddings	55,979	$334,521,873.69

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. The second query took approximately 9 seconds to process, much longer than the previous 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
```



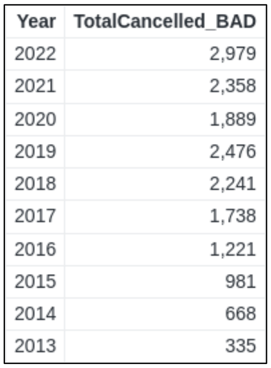
Category	TotalLoansByCategory	TotalAmount
Car Loan	785,882	$22,968,330,202.57
Consolidation	956,978	$19,611,447,243.03
Credit Card	5,944,363	$29,836,636,910.85
Education	448,927	$16,710,867,033.57
Home Improvement	672,629	$4,156,432,651.00
Major Purchase	112,125	$669,329,565.87
Medical	336,313	$12,097,395,998.00
Mortgage	855,366	$318,340,498,613.16
Moving Expenses	55,744	$333,383,431.31
Personal	111,949	$668,522,275.42
Small Business	783,950	$15,849,401,780.73
Vacation	112,216	$672,678,516.20
Weddings	55,979	$334,521,873.69

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 native Oracle. Instead, it wrote a query to bring only the necessary columns to the SAS Compute Server for processing. This query took approximately six seconds to execute.

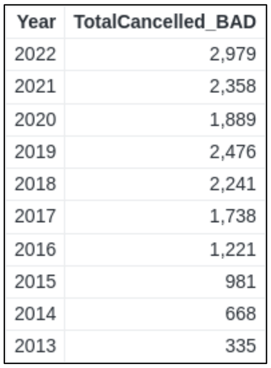
```
...
SAS_SQL:  Unable to convert the query to a DBMS specific SQL statement due to an error.
...
SQL_IP_TRACE: Some of the SQL was directly passed to the DBMS.

ORACLE_24: Prepared: on connection 0
SELECT  "Year", "CancelledReason" FROM STUDENT.LOANS_RAW


ORACLE_25: Executed: on connection 0
SELECT statement  ORACLE_24
```



Year	TotalCancelled_BAD
2022	2,979
2021	2,358
2020	1,889
2019	2,476
2018	2,241
2017	1,738
2016	1,221
2015	981
2014	668
2013	335

View the log for the second query. Notice that the SQL procedure syntax was converted into native 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. This query ran in approximately two seconds, in about a third of the time that it took the previous query to run.

```
...
ORACLE_2: 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

SQL_IP_TRACE: pushdown attempt # 1
SQL_IP_TRACE: passed down query:    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
SQL_IP_TRACE: The SELECT statement was passed to the DBMS.

ORACLE_3: Executed: on connection 0
SELECT statement  ORACLE_2

ACCESS ENGINE:  SQL statement was passed to the DBMS for fetching data.
```



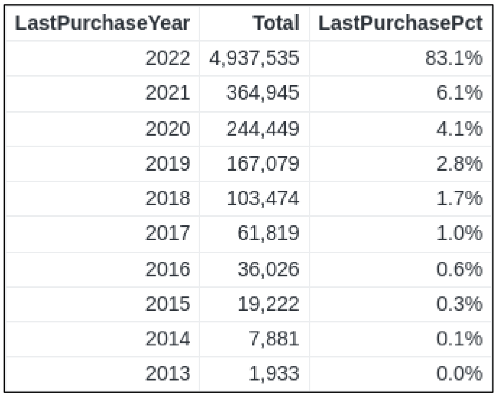
Year	TotalCancelled_BAD
2022	2,979
2021	2,358
2020	1,889
2019	2,476
2018	2,241
2017	1,738
2016	1,221
2015	981
2014	668
2013	335

e. The last query in this section uses the SAS YEAR and DATEPART functions to summarize the year of the **LastPurchase** date column to count the last time that a credit card was used for each account by **Year**.

```
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 native database SQL due to an error. However, implicit pass-through converted the subquery (ORACLE_10) to native Oracle SQL to run in the database. It also converted the main query (ORACLE_11) to select only the necessary columns (**LastPurchase** and **Category**) and rows (*Credit Card*) to bring back to the SAS Compute Server for processing. This query took approximately nine seconds to run.

```
SAS_SQL:  Unable to convert the query to a DBMS specific SQL statement due to an error.
SQL_IP_TRACE: pushdown attempt # 1
SQL_IP_TRACE: passed down query:    select DATEPART
ACCESS ENGINE:  SQL statement was not passed to the DBMS, SAS will do the processing.
...

ORACLE_10: Prepared: on connection 0
 select COUNT(*) from STUDENT.LOANS_RAW TXT_2 where TXT_2."Category" = 'Credit Card'

SQL_IP_TRACE: passed down query:    select COUNT(*) from STUDENT.LOANS_RAW TXT_2 where
TXT_2."Category" = 'Credit Card'
SQL_IP_TRACE: Some of the SQL was directly passed to the DBMS.

ORACLE_11: Prepared: on connection 0
SELECT  "LastPurchase", "Category" FROM STUDENT.LOANS_RAW  WHERE  ("Category" = 'Credit
Card' )


ORACLE_12: Executed: on connection 0
SELECT statement  ORACLE_11


ORACLE_13: Executed: on connection 0
SELECT statement  ORACLE_10

ACCESS ENGINE:  SQL statement was passed to the DBMS for fetching data.
```



LastPurchaseYear	Total	LastPurchasePct
2022	4,937,535	83.1%
2021	364,945	6.1%
2020	244,449	4.1%
2019	167,079	2.8%
2018	103,474	1.7%
2017	61,819	1.0%
2016	36,026	0.6%
2015	19,222	0.3%
2014	7,881	0.1%
2013	1,933	0.0%

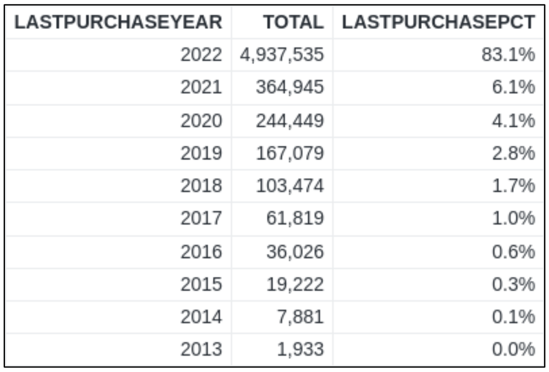
4. In the EXPLICIT PASS-THROUGH ON THE SAS COMPUTE SERVER section, we will use explicit pass-through to write and submit native Oracle SQL using the SQL procedure. This enables us to use native database features, and it will ensure that the query runs inside the database.

a. The following query uses native Oracle SQL (explicit pass-through) to achieve the same results as the previous 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 native Oracle SQL query directly to the database for processing. This explicit pass-through query ran in approximately three seconds, or one-third of the time as the implicit pass-through query.

```
ORACLE_35: 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_36: Executed: on connection 2
SELECT statement  ORACLE_35
```



LASTPURCHASEYEAR	TOTAL	LASTPURCHASEPCT
2022	4,937,535	83.1%
2021	364,945	6.1%
2020	244,449	4.1%
2019	167,079	2.8%
2018	103,474	1.7%
2017	61,819	1.0%
2016	36,026	0.6%
2015	19,222	0.3%
2014	7,881	0.1%
2013	1,933	0.0%

CAS Server Database Processing

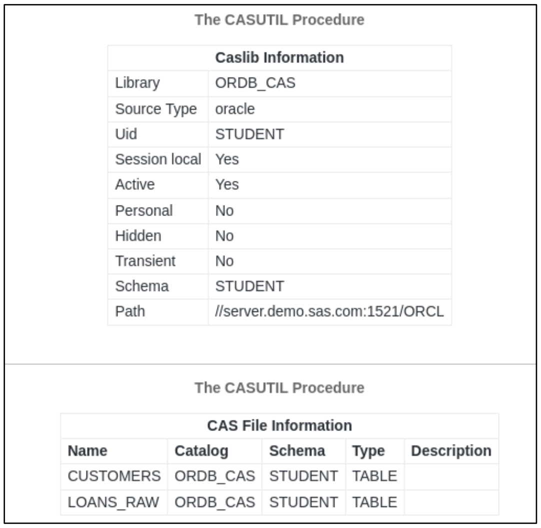
1. Open the **02 – CAS Server.sas** program. The CAS statement makes a connection to the CAS server from the Compute server. The CASLIB statement creates a CAS server connection to the same Oracle database that we used earlier. Run the CAS and CASLIB statements

```
cas conn;

caslib ordb_cas datasource=(srctype="oracle",
                            user="STUDENT",
                            password="Metadata0",
                            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.

```
proc casutil incaslib = 'ordb_cas';
   list files;
   list tables;
quit;
```

View the results. 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
Uid	STUDENT
Session local	Yes
Active	Yes
Personal	No
Hidden	No
Transient	No
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	

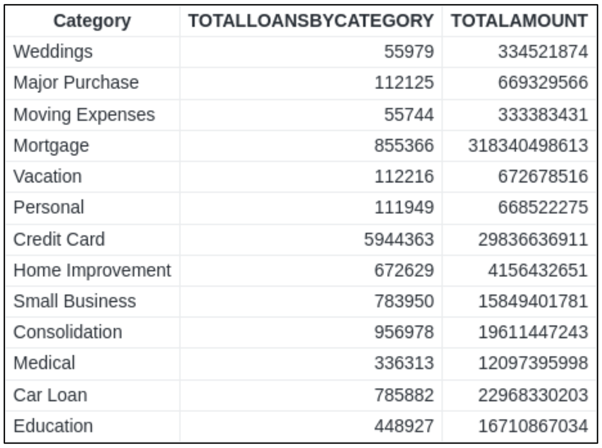
```
NOTE: No tables are available in caslib ORDB_CAS of Cloud Analytic Services.
```

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 to specify our CAS connection name instead of the SQL procedure. You must reference the database table name, not a CAS table for implicit pass-through to work. Run the FEDSQL procedure.

```
proc fedsql sessref=conn _method;
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.

```
...
Offloaded SQL statement
------------------------


        select "STUDENT"."LOANS_RAW"."Category", COUNT ( * )  as "TOTALLOANSBYCATEGORY", SUM
("STUDENT"."LOANS_RAW"."Amount")  as
"TOTALAMOUNT" from "STUDENT"."LOANS_RAW" group by "STUDENT"."LOANS_RAW"."Category"

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



Category	TOTALLOANSBYCATEGORY	TOTALAMOUNT
Weddings	55979	334521874
Major Purchase	112125	669329566
Moving Expenses	55744	333383431
Mortgage	855366	318340498613
Vacation	112216	672678516
Personal	111949	668522275
Credit Card	5944363	29836636911
Home Improvement	672629	4156432651
Small Business	783950	15849401781
Consolidation	956978	19611447243
Medical	336313	12097395998
Car Loan	785882	22968330203
Education	448927	16710867034

4. You can also use explicit pass-through with a caslib. In this example, we will use explicit pass-through to subset the database table using Oracle SQL. Then the SAS SELECT statement creates an in-memory distributed CAS table with the results from Oracle. Run the FEDSQL procedure.

```
    NOTE: Table CCACCOUNTS2022 was created in caslib ORDB_CAS with 4937535 rows returned.
```

5. In the Analyze a CAS table section, we will process an in-memory CAS table using a variety of methods.

    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;
```

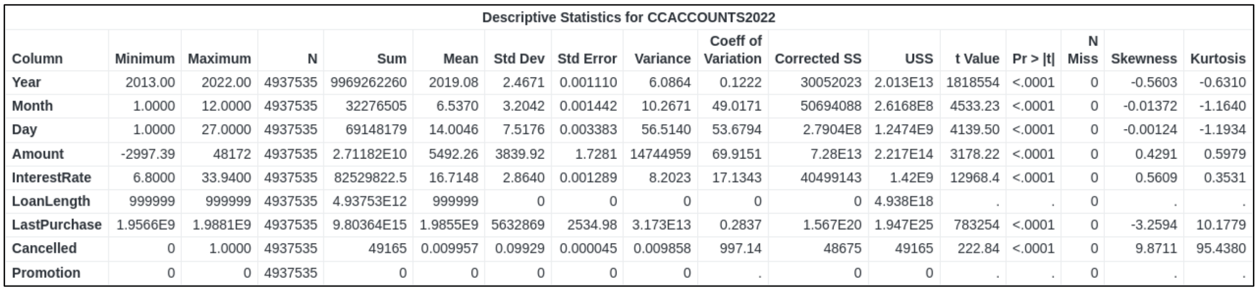
b. Once a file is loaded into CAS, you can begin processing the in-memory table. You can use the FEDSQL procedure with the SESSREF= option to execute SQL queries on a CAS table. Here, we run two simple queries on the CAS table to process the data in the distributed CAS server. Run the FEDSQL procedure.

```
proc fedsql sessref=conn _method;
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;
```

c. Once a table is loaded into memory, you can also execute native CAS actions to process the data. For example, you 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 results. Notice that the summary action is processed in about under half a second and returns a variety of descriptive statistics. Imagine writing this query!



Descriptive Statistics for CCACCOUNTS2022

									Coeff of					N		
Column	Minimum	Maximum	N	Sum	Mean	Std Dev	Std Error	Variance	Variation	Corrected SS	USS	t Value	Pr > |t|	Miss	Skewness	Kurtosis
Year	2013.00	2022.00	4937535	9969262260	2019.08	2.4671	0.001110	6.0864	0.1222	30052023	2.013E13	1818554	<.0001	0	-0.5603	-0.6310
Month	1.0000	12.0000	4937535	32276505	6.5370	3.2042	0.001442	10.2671	49.0171	50694088	2.6168E8	4533.23	<.0001	0	-0.01372	-1.1640
Day	1.0000	27.0000	4937535	69148179	14.0046	7.5176	0.003383	56.5140	53.6794	2.7904E8	1.2474E9	4139.50	<.0001	0	-0.00124	-1.1934
Amount	-2997.39	48172	4937535	2.71182E10	5492.26	3839.92	1.7281	14744959	69.9151	7.28E13	2.217E14	3178.22	<.0001	0	0.4291	0.5979
InterestRate	6.8000	33.9400	4937535	82529822.5	16.7148	2.8640	0.001289	8.2023	17.1343	40499143	1.42E9	12968.4	<.0001	0	0.5609	0.3531
LoanLength	999999	999999	4937535	4.93753E12	999999	0	0	0	0	0	4.938E18	.	.	0	.	.
LastPurchase	1.9566E9	1.9881E9	4937535	9.80364E15	1.9855E9	5632869	2534.98	3.173E13	0.2837	1.567E20	1.947E25	783254	<.0001	0	-3.2594	10.1779
Cancelled	0	1.0000	4937535	49165	0.009957	0.09929	0.000045	0.009858	997.14	48675	49165	222.84	<.0001	0	9.8711	95.4380
Promotion	0	0	4937535	0	0	0	0	0	.	0	0	.	.	0	.	.

6. One common mistake with users is that they use PROC SQL to try to run queries on an in-memory CAS table. To execute PROC SQL on a CAS table, you first need to create a library reference to a caslib with the CAS engine using the LIBNAME statement. Then you can use that library reference to try to push SAS code to the CAS server. If the SAS code is not CAS enabled, the CAS table is transferred to the Compute Server for processing. This works similarly to implicit pass-through to a database. 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;
```

View the log. Notice that you received an error because the maximum allowed data size being transferred from CAS to the Compute Server is larger than the default value of the DATALIMIT option.

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